College of Engineering, Design and Physical Sciences

Department of Electronic and Electrical Engineering

BEng. Computer Systems Engineering

Brunel University of London

FHEQ Level 6 BEng Final Year Project Report

NON-INTRUSIVE OCCUPANCY DETECTION IN UNIVERSITY SPACES USING EDGE AI AND TIME-SERIES SENSOR DATA

Brunel University of London,

Uxbridge, Middlesex, United Kingdom

Member of IEEE

<Anonymous Copy>

April 14th

Abstract

Abstract- Indoor environmental quality is a critical factor influencing building occupants' health, comfort, and productivity. Naturally occurring elements such as temperature, humidity, air quality, and occupancy density play direct roles in determining the overall well-being of individuals within shared spaces. Poor indoor conditions can lead to health risks, reduced performance, and increased energy costs. With the growing global emphasis on sustainability and occupant-centric building designs, real-time monitoring of environmental variables has gained considerable attention.

Modern sensor technologies can continuously track indoor environmental conditions, detect anomalies, and provide valuable insights for more innovative building management. By integrating sensor-based monitoring systems, facility managers and occupants can better understand how environmental factors fluctuate and respond to changing usage patterns. That forms the basis for creating healthier, more efficient, and adaptable indoor environments.

This project focuses on developing a basic environmental and occupancy monitoring system by collecting and analysing real-world data from student accommodation and university study spaces. By deploying various sensors, the system captures temperature variations, humidity levels, air quality (specifically particulate matter concentrations), and occupancy rates across different rooms and settings. This approach comprehensively explains how different factors interact within frequently used environments. The collected data serves as a baseline for evaluating environmental quality and lays the groundwork for potential future applications such as predictive analytics, energy optimisation, and proactive indoor air management.

Keywords—EdgeAI, TinyML, CNN

Significance

The significance of this project stems from its practical, real-world application in commonly occupied spaces. Student accommodations, postgraduate offices, and library study rooms are high-traffic areas where optimal indoor conditions are essential. Good indoor air quality and stable thermal environments

directly contribute to the occupants' cognitive function, physical health, and overall comfort. Moreover, environmental quality can influence learning efficiency, collaboration, and student well-being in university settings.

The project highlights how human activity impacts room conditions and resource usage by monitoring environmental variables and tracking occupancy levels. Understanding these patterns is vital for improving ventilation strategies, adjusting heating and cooling systems, and planning maintenance activities more efficiently. Detecting abnormal environmental changes early can prevent potential health risks, such as exposure to high particulate matter levels or poor ventilation, which could otherwise go unnoticed.

While the current project scope focuses primarily on data collection and analysis, it provides a solid foundation for future work. The gathered data can support the development of lightweight machine learning models (TinyML) to automatically predict occupancy trends or identify risks. Additionally, the project aligns with broader sustainability initiatives by demonstrating the benefits of systematic environmental monitoring. Optimising building operations based on real-time conditions can significantly contribute to energy conservation, cost savings, and achieving net-zero carbon targets.

ACKNOWLEDGEMENTS

I would like to dedicate this paper to my Mother, my Father and the rest of my amazing family who has not only supported me in my toughest moments – but also aided with mental health/workload across my entire academic journey and my career, alongside equipping me with valuable life skills, and provided motivation across my journey.

I would also like to personally acknowledge and recognize how much my Supervisor, has supported and had faith in taking me on as a Supervisor, and to let me take on this project in Edge AI.

I would like to personally thank Dr. Konstantinos Banitsas, and Dr. Ian Dear for supporting, and being very understanding of my circumstances that I have been presented with upon starting my academic journey in 2023.

I would like to personally acknowledge the Department of Electronic and Electrical Engineering, as it has not only challenged me but given me the opportunity to pursue my goals – and assist with developing my skills and knowledge by providing me with the resources, and valuable modules enabling me to push for further learning and development in my skills, and most importantly depth in knowledge.

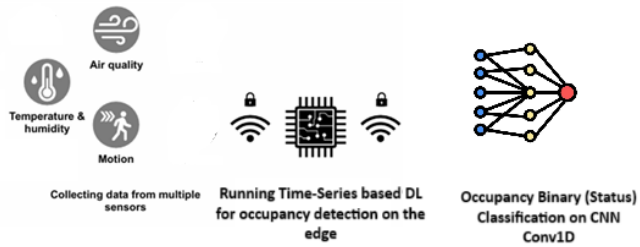
TABLE OF CONTENTS:

I. INTRODUCTION

A. Aims & Objectives

1. Aim

This project aimed to develop a basic environmental and occupancy monitoring system by collecting and analysing sensor data from student accommodation and university study spaces. The goal is to observe how environmental conditions and occupancy levels vary over time and space, providing insights into indoor environmental management.



2. Objectives

The primary aim of the environmental monitoring system is to not only to monitor the trending, and noticeable (anomalous) patterns in temperature fluctuations specifically in different areas of the room/offices of a specific building. However, it will additionally use the data collected from running the hardware-implemented system to train a lightweight Machine Learning model **(TinyML)** for predicting patterns and thresholds that allow predicting occupancy levels based on using multi-modal classification using (insert here). This will be designed to be implemented as a risk management tool, as opposed to simply monitoring environmental variable readings, temperature/humidity, and particle matter/gases without any conclusions or outputs.

The project should be designed to prevent negative side effects to the respiratory health associated with the atmosphere present in, different office spaces. Also, to prevent any negative stimuli or unsuitable environment for boosted productivity of the students/colleagues spending time and being affected by the respective office spaces in the project.

By using sustainable, and renewable sources, it can help save costs in utilities, in particularly the energy consumption (electricity) used in the Office Spaces. Therefore, meeting the "NET-ZERO" goals that are established by the Government, to sustainability goals. Hence, this can act as a "eco-friendly", and "low carbon", in comparison to the IAQ Monitoring systems being distributed in today's market.

To achieve the aims outlined (above) of the final year project, several stages and requirements are to have been met, interdependently. The stages are known as research, design, implementation, experimental method and testing, methodologies, and a conclusion. They will need to be conducted not only to validate, otherwise also to produce qualitative research based on comprehensive published journals/conference papers and open-source market report samples (not full version), including evaluating several research papers from various reputable sources e.g. ieeexplore.com.

II. PROBLEM

By utilizing principles based on the Cloud Computing Framework, called "Edge Computing", we can reduce debilitating network latency and increase the capacity of bandwidth communication between the hardware sensors (Edge nodes) and the Processing Unit, since it is not offloaded to a

Cloud Environment e.g. Azure, or Kubernetes. Contrary to this approach, it utilizes the hardware and networking capabilities of the Processing Unit, so the definition of "Edge" is not to be confused with Multiaccess Edge Linux.

Based on the market research reports and forecast of upcoming technological trends, there are a wide variety of" Smart" Indoor Air Quality monitoring systems which also provide a vigilance and built-in security system, and identify (*Edge Computing Market-global Forecast to 2029 EDGE COMPUTING MARKET GLOBAL FORECAST TO 2029, 2023)*

Employing a minimal dataset strategy will streamline sensor data collected from the MLX90640, DHT22, and MQ Gas sensors, by compressing the datasets using Data Compression techniques such as using Bayesian Classifier, Auto encoding, and other types of techniques. This allows making it faster for the models to be trained, allowing for lower processing requirements while still maintaining high performance for machine learning applications, since it doesn't require an accelerated.

Project Management

Potential Risks

The potential risks in associated with AI, are closely related to the unethical uses of data either obtained with consent, or without consent ().

The potential consequences of mis-using data collected through sensors, like environmental variables, or compromising information obtained through cookies, and databases can vary from moderate to extremely dangerous. where it is designed to complete tasks that may otherwise be used for specifical classification/regression tasks that may break the code of conducts, in e.g. universities/schools and in extreme cases the law in the United Kingdom.

Accuracy and Biases are also a very important factor in determining the reliability and calculated accuracies of using the models are also very important – since this can potentially be very determining factor especially when decisions are made that will directly affect the wellbeing, or mechanical process for instance, (PID controller).

Risks Mitigation

To mitigate the risks, in relation to the applications of AI, and unethical uses of Artificial Intelligence would be the introduction and regularization standards introduced by the BS EN standards outlined by the Government. By doing so, it is a set of regulations that team of developers and other departments

are required to follow to ensure **safety, reliability, and performance** in AI and engineering applications, **please refer to the Appendix for the BS EN Compliance Laws and Regulations**.

Gantts Chart

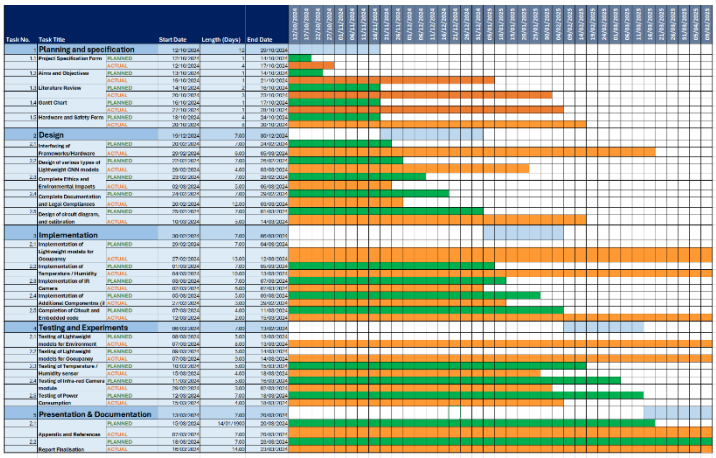


Figure A - Gantts Chart (larger resolution in List of Tables)

III. LITERATURE REVIEW

The system incorporates lightweight deep learning models to TinyML and lightweight model papers) in order to detect patterns/sequences that are affected by contextual AI based on quantitative data obtained organically from environmental conditions based on testing/experimentation from using analogue/digital sensors inside an office environment, including various large open-source datasets.

Using the information based on the datasheets collected in the process of finding the most optimal edge devices currently in the market (relative to a budget) – a common re-occurrence of specific brands/devices kept re-appearing i.e. Raspberry Pi, ESP32, and STM32. (Easterline et al., 2024)

Specifically, the ESP32, and the ESP8266 microcontroller is commonly referenced and highly recommended, hence implemented as an MCU for data collection. Due to its high versatility, and flexible approach due to its support, and high compatibility with many sensor equipment currently employed in the current industry, especially when monitoring ambient temperatures of industrial settings, or for

smart homes as the specific microcontroller, and use of Gas Sensors are implemented in the monitoring system based on the investigation completed by (Torfs et al., 2013)

Although, this is typically quoted as the main use for various real-time applications and interfacing of sensors, particularly in the discussion of using wireless networks and protocols such as LoRoWAN.(Easterline et al., 2024)

On the contrary, the trade-off for sacrificing the threshold of the i2c protocol connecting the Raspberry Pi to the temperature/humidity sensors, can equivalently account for the machine learning algorithms taking place inside the Linux environment.(Dahane, Benameur and Kechar, 2022; Raspberry Pi 5, 2025)

The problems in relation to the quality of the Indoor Air Quality, in relation to indoor environments and spaces, differ mostly from a range of different types of building/habitats and the technology installed in relation to the office spaces, e.g. Smart Buildings, and University Study area.

In particularly, in developing countries such as India, are greatly affected by the high concentration of harmful particle matters/gases in the outdoor air quality.

Table 1

Author/Year	Sampling Strategy	Research Paper Title	Data Collected / Application
Mainwaring et al. (2002)	Periodic/Continuous Sampling	*Wireless Sensor Networks for Habitat Monitoring*	Environmental data (e.g., temperature, humidity)
Krishnamachari et al. (2002)	Adaptive Sampling / Data Aggregation	*The Impact of Data Aggregation in Wireless Sensor Networks*	Environmental sensor data (temperature, etc.)
Akkaya & Younis (2005)	Survey (including Sampling Strategies)	*A Survey on Routing Protocols for Wireless Sensor Networks*	Overview of environmental monitoring and data collection
Yick, Mukherjee & Ghosal (2008)	Adaptive Sampling	*Wireless Sensor Network Survey*	General sensor data in wireless sensor networks
Bengio et al. (2015)	Scheduled Sampling	*Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks*	Sequence data for language modelling (illustrative method)

On the topics, of collecting data – an alternative approach to sampling datasets time

We can use Few-Shot Learning to help train and improve fine-tuning of ML models on embedded systems running on the Edge, such as Threshold-Based Alerts, or Scheduled Sampling to reduce the

time taken to train the models, in particularly when the datasets don't accurately mimic the behaviour of the data incoming from the sensors used in the project.

This approach is particularly useful when you have limited data or when data collection is expensive or time-consuming.

Another problem, is related to the amount of power/voltage that is required to supply the data collection layer, if Real-Time Processing is required. In this case, an alternative cost-effective approach would be to implement "Scheduled Sampling" into the data collection later. It is an approach that would typically be recommended for collecting data between different moments inside a time-frame (extended), as opposed to collecting data in real-time which may present future problems. In particularly when discussing about the electricity required (power supply) to run the sensors for long periods of time, and creating more troubleshooting issues (hardware-wise), by trying to gain accurate data.(Bengio et al., no date)

Table 2

Author / Year	Problem	Size of Dataset (Data)	Methodology	# of Datapoints	Results	Occupancy
Steinemann / 2023	Exposure to fragranced consumer products impacting air quality	Survey data on consumer products and VOC emissions	Analysed VOC emissions from consumer products and their impact on IAQ.	Survey-based dataset	Identified that fragranced products, including those labelled as "green" or "organic," emit hazardous air pollutants, impairing indoor air quality and posing health risks.	Yes
Health.com / 2024	Indoor air pollution caused by certain cooking methods	Particulate matter (PM) and VOC emissions from different cooking methods	Monitored particulate matter and VOCs released during various cooking techniques.	Multiple cooking scenarios monitored	Determined that air frying produces significantly lower levels of indoor air pollutants compared to pan frying and deep frying, thus improving indoor air quality.	Yes
Arxiv.org / 2024	Inadequate ventilation in educational settings affecting air quality	CO_2 and IAQ measurements from classrooms	Measured CO_2 and indoor pollutants in classrooms to evaluate ventilation adequacy.	CO_2 and PM data from schools	Confirmed that poor ventilation in classrooms leads to elevated CO_2 levels, adversely affecting students' cognitive functions and overall health.	Yes
Arxiv.org / 2023	Indoor air pollution from human activities and building design	Air quality measurements from various indoor environments	Conducted a field study using custom-built air quality sensors to monitor IAQ dynamics across different indoor spaces.	Data from over 28 indoor spaces	Revealed that room structures, ventilation systems, and occupant activities significantly impact the accumulation and spread of indoor air pollutants, emphasizing the need for comprehensive IAQ monitoring.	Yes
Arxiv.org / 2021	Impact of human interaction and building design on indoor air quality	Simulated data based on building layouts and occupant behaviours	Developed an agent-based simulator to estimate adequate room sizes, ventilation parameters, and policies affecting IAQ.	Simulation data from various building scenarios	Demonstrated that human-building interaction patterns significantly influence IAQ, and that advanced simulation tools can aid in designing healthier indoor environments.	Yes
Arxiv.org / 2024	Correlation between outdoor and indoor PM2.5 levels	Indoor and outdoor PM2.5 measurements across multiple buildings	Predicted hourly indoor PM2.5 concentrations and investigated their relationship with outdoor levels using machine learning models.	Data from 91 monitoring sensors in 24 buildings	Found that outdoor PM2.5 concentrations significantly impact indoor air quality, especially during events like bushfires, highlighting the importance of accurate indoor air quality prediction for public health.	Yes

Particularly in emerging areas where limited access to HVAC systems and increasing combustion contribute to poor air conditions, this research addresses numerous important difficulties connected to monitoring Indoor Air Quality (IAQ). Problems like erroneous temperature readings from low-cost sensors, crowded areas influencing IAQ (particularly in Australia, London, and India), and energy

inefficiencies resulting from unequal airflow distribution are under focus. Providing a reasonably priced substitute for real-time monitoring, the solution seeks to lessen reliance on costly commercial-grade sensors and proprietary technologies like ZigBee. Furthermore, suggested are reasonably priced occupant detection and machine learning integration, therefore enhancing privacy and scalability without requiring expensive AI modules or limited-service subscriptions.

Table 3

Author/Year	Dataset Name	Record Size	# Classes	Class Balance
Ardeshir Mahdavi, BergerBerger, Farhang TahmasebiFarhang Tahmasebi, Matthias Schuss/2013	01_occ.CSV	35,040	101	Class 0.0 has 26,542 examples. Class 1.0 has 519 examples. Class 0.09 has 388 examples.
PallaviPatia/2015	Occupancy.CSV	20,560	2	Class 0 has 15,810 examples. Class 1 has 4,750 examples.
S. De Vito, E. Massera, M. Piga, L. Martinotto, G. Di Francia/2018	AirQuality.CSV	9,471	x	x
De Vito et al / 2008	AirQualityUCI.CSV	9,471	4	Class low: 5129, Class medium: 2163, Class high: 382, Class unknown: 1797

Table 4

Author/Year	Dataset Name	Class 1	Class 2	Class 3	Class 4
PallaviPatia / 2015	Occupancy (2015)	Vacant	Occupied		
Ardeshir Mahdavi, Berger, Farhang Tahmasebi, Matthias Schuss / 2013	01_occ (2013)	Vacant (0.0)	Occupied (1.0)	Intermediate values (e.g.	0.09
S. De Vito, E. Massera, M. Piga, L. Martinotto, G. Di Francia / 2018	Air Quality (2018)	CO level (continuous)	NOx level (continuous)	Benzene level (continuous)	Temperature
De Vito et al. / 2008	AirQualityUCI (2008)	Low CO levels	Medium CO levels	High CO levels	Unknown CO levels
Szeged, Hungary / 2006-2016	weather History (2006-2016)	Partly Cloudy	Mostly Cloudy	Overcast	

The classes that have been used in the chosen dataset/s, were Vacancy Levels, Occupancy Levels, CO levels, NO Levels, Temperature, and Humidity. In this case, the vacancy levels contributed towards the training the machine learning models

In occupancy monitoring systems, several data representations have been widely adopted to effectively capture and process environmental and human presence signals. Time-series tabular data, typically sourced from sensors such as temperature, humidity, CO_2, and PIR, allows for structured sequential analysis and integration with models like Conv1D or traditional machine learning algorithms. Thermal infrared grids, such as those from the MLX90640 sensor, offer low-resolution spatial heat maps that can be processed as 2D arrays or flattened for compatibility with convolutional models, enabling privacy-preserving human detection. Binary presence flags provide lightweight indicators of occupancy states, often used in threshold-based systems or as inputs for classification models. Additionally, sliding window representations are used to segment temporal data into fixed-length sequences, capturing dynamic occupancy patterns over time. These formats enable efficient feature extraction, temporal analysis, and edge-compatible deployment, making them central to modern occupancy detection approaches.

Table 5

Author/Year	Dataset	Rep. 1	Rep. 2	Rep. 3
Ardeshir Mahdavi, Berger, Farhang Tahmasebi, Matthias Schuss / 2013	01_occ	"Vacant" or "Occupied" status	Timestamps	Room occupancy patterns
PallaviPatia / 2015	Occupancy	"Vacant" or "Occupied" status	Timestamps	Correlation with CO2 and tempe
S. De Vito, E. Massera, M. Piga, L. Martinotto, G. Di Francia / 2018	Air Quality	Environmental sensor readings (e.g., CO levels)	Temporal data	Pollutant concentration trends
De Vito et al. / 2008	AirQualityUCI	CO levels categorized as "Low," "Medium," "High"	Temporal pollutant changes	
Szeged, Hungary / 2006-2016	weather History	Weather condition summaries (e.g., "Clear," "Foggy")	Temporal trends (timestamps)	Weather attribute variations

To explore the data in a more balanced approach, we decided to use several data augmentation techniques including the type of data augmented and the quantized neural networks.

In this case, data augmentation occurred on datasets and real-time data extracted from data collection, as an attempt to compress the data in order to allow the CNN model to train using the data. Data compression in this context serves to reduce the dimensionality and size of the input while preserving critical features necessary for accurate classification. This enables faster training times, lower memory consumption, and more efficient use of computational resources—particularly important for deployment on edge devices with limited processing capabilities (Metwaly et al., 2019). Additionally, effective compression techniques can suppress irrelevant noise within the dataset, potentially improving

the model's generalization ability (Mohammadabadi et al., 2022). However, it is essential to strike a balance; overly aggressive compression may lead to information loss and decreased model performance. Therefore, compression and augmentation were used here not only to optimize the input for CNN-based training, but also to maintain sufficient data richness for accurate occupancy detection.

Of which, we were able to use TinyML-specific frameworks and hierarchical feature extraction from the infrared thermal camera sensor readings. In this case, the raw thermal data captured by the MLX90640 infrared array sensor—comprising a 32×24 grid of temperature values—was converted into structured input suitable for machine learning. Each frame provided a low-resolution thermal map of the scene, allowing the system to detect human presence based on heat signatures without capturing any identifiable visual information, thus preserving user privacy. The hierarchical feature extraction process involved normalization, resizing, and, in some cases, flattening or transformation into time-series sequences, enabling compatibility with Conv1D or lightweight CNN architectures. These preprocessed inputs were then fed into the training pipeline within TinyML frameworks such as TensorFlow Lite, allowing the model to learn meaningful spatial and temporal patterns from compact, edge-suitable thermal data. This process ultimately supported real-time, on-device occupancy detection with minimal computational overhead and strong privacy guarantees.

Convolutional Neural Networks (CNNs) designed for classification tasks typically consist of an upstream feature extractor followed by a downstream classifier. This architecture is particularly suitable for processing time-series data when implemented using one-dimensional convolutional layers (Conv1D), which are effective at capturing temporal dependencies. According to Zhang et al. (2024), Smart Health Monitoring systems have successfully adopted lightweight CNN architectures such as MobileNetV2, which operate using Edge Computing and On-Device Computing principles. These models demonstrate similar computational strategies, enabling efficient inference directly on embedded hardware. The primary benefits of this approach include reduced reliance on cloud resources, lower system costs, and the ability to deploy complex models locally while maintaining real-time performance. In recent edge AI applications, a diverse range of machine learning and deep learning models have been employed to enable real-time, efficient processing in domains such as smart agriculture, health monitoring, and surveillance. Commonly used models include classical algorithms like Support Vector Machines (SVMs) and Random Forests, which offer fast inference and low resource usage but may lack scalability or robustness in complex data environments (Yang et al., 2024; Rumy et al., 2021). In contrast, deep learning models such as Convolutional Neural Networks (CNNs), Long Short-Term

Memory (LSTM) networks, and Recurrent Neural Networks (RNNs) are increasingly favored due to their ability to capture spatial and temporal patterns in sensor and image data (Proietti et al., 2021; Fathoni et al., 2024). CNNs, in particular, are widely used in occupancy and health monitoring systems, with 1D convolutions (Conv1D) proving effective for time-series sensor data (Hassan et al., 2022). Lightweight variants such as MobileNetV2 have been adopted for on-device computing to reduce reliance on cloud services while supporting real-time inference (Zhang et al., 2024). More complex approaches, including Reinforcement Learning and Boosted Decision Trees, have also been proposed to enhance safety and adaptability, though they often require further validation or incur deployment challenges (Shafi et al., 2023; Zhang et al., 2024). Overall, the selection of AI models reflects a trade-off between accuracy, resource efficiency, and deployment complexity in edge computing environments.

In this case, according to (David et al., 2021; Pedregosa et al., 2011; Abadi et al., 2016; Paszke et al., 2019, a wide range of programming environments and tools are employed for AI model development and deployment, each tailored to specific system requirements and application domains. Python remains the dominant language due to its extensive ecosystem of machine learning libraries such as PyTorch, TensorFlow, Scikit-Learn, and TensorFlow Lite, which support both general-purpose model training and lightweight, edge-oriented deployment. For large-scale and distributed processing, Java-based environments like Apache Spark MLlib and Deeplearning4j are used to handle high-volume data across cloud-edge systems (Meng et al., 2016; Gibson et al., 2016). Meanwhile, Rust is emerging as a systems-level alternative with libraries like Linfa and SmartCore, offering memory safety and performance benefits for embedded ML (Yaw, 2023; Orlov, 2020). On low-level embedded platforms, Embedded C remains standard, with environments such as Keil uVision and MPLAB X supporting direct hardware control on microcontrollers (Arm Ltd., 2017; Microchip, 2016). Finally, C++ is frequently used in real-time computer vision applications through libraries like OpenCV and dlib, or for integrating TensorFlow models into GUI-based tools via Qt (Bradski, 2000; Singh et al., 2020). Together, these tools reflect a layered ecosystem—ranging from high-level model training to low-level embedded deployment—driven by performance, memory constraints, and real-time requirements.

To calculate bandwidth of network latency, we will be analysing, throughput, and various other performance metrics used in Wireshark (and networking applications)

In order to make an impact environmentally, using AI – we will be exploring the different literature associated with environmental awareness and impact of running Machine Learning/Artificial Intelligence models as it requires electricity.

Unlike the research based on Quantum Computing, Edge Computing still heavily relies on the traditional computer processor/graphic card units to be not only be trained, however be deployment for prototyping purposes or running databases such as ChatGPT

Based on the research, conducted by (REFERENCE), it is evident the unit of measurement is the carbon footprint.

In order to measure this, we will be using a library in python called the

In addition to this, we will monitoring power consumption bey placing sensos that will directly analyse the electrical/electronic signals and generate a score based on a benchmarking system, that relies on other power computing data of other microcontroller computing units.

Calculating the expense rate of specific AI models involves assessing their computational demands, model size, memory footprint, and inference latency. In this project, training and optimization were conducted remotely using Lightning AI, with the final model converted to TensorFlow Lite for deployment on a Raspberry Pi 5. While direct hardware-level profiling during training was not feasible due to cloud abstraction, performance metrics such as training duration, model size, and conversion time were recorded. For evaluation on the edge device, on-device inference time, CPU usage, and memory consumption were measured to estimate runtime efficiency and validate the model's suitability for low-power embedded deployment.

Table 6

Author/Year	Name of Hardware	Price of Hardware	Energy Efficiency (Watts)	Open-Source Availability	Price of Software Required
Pearce et al., 2021	RepRap 3D Printer	~$500 USD	~60 W	Fully Open Source	Free (RepRap Firmware)
Chagas et al., 2022	Arduino-based Tools	$20–$50 USD	~10–15 W	Fully Open Source	Free (Arduino IDE)
Tiwari & Yadav, 2018	Raspberry Pi	~$35 USD	~5 W	Partially Open Source	Free (Raspbian OS, Python IDE)
Smith et al., 2019	Mixed Lab Equipment	Varies (low-cost mix)	Varies (low-resource settings)	Partially Open Source	Free or Proprietary (LabVIEW: ~$500 USD

To make the project more cos-effective we have opted to not only use cheaper material, and sourced them from AliExpress (Distributor based in China), but we have also decided to develop the program using not only open-source but free software in order to allow us to create, load, train the model and obtain open source datasets.

In other words, this has made the this project more sustainable, and reliable on a long-term basis, rather than using nationwide distributors such as Amazon, eBay.

Table 7

Sensors/Equipment	Quantity	Price Paid (Feb 2025)	Total Paid	Current Price (Apr 2025)	Total Current Price	Price Difference
SDS011 PM2.5 Laser Sensor	1	£13.19	£22.96	£23.95	£23.95	£0.99
DHT22 AM2302 Sensor Module	4	£0.78	£6.64	£4.59	£18.36	£11.72
MQ-135 Air Quality Sensor Module	4	£3.62	£17.38	£6.00	£24.00	£6.62
MLX90640 Thermal Camera Module	3	£29.69	£106.88	£70.03	£210.09	£103.21
STM32H743IIT6 Core Development Board	1	£16.69	£16.69	£47.03	£47.03	£30.34

Building environmentally friendly, green AI systems isn't an issue according to (Tschand et al., 2024), it is evident that the ability to measure the energy efficiency of different machine learning algorithms in respect to requiring technical in-depth knowledge/experience isn't required to do so. Different techniques, e.g. are employed especially when manufacturing and designing AI accelerated hardware to run Neural Networks and Tensors more efficiently, saving energy and money simultaneously. ("Energy-efficient AI hardware design for edge intelligence," 2024)

Since the system will utilize different layers of networking architecture, the ontology will be deployed/distributed across different layers of ontology. In this case, since are not restricted to one type of technology/use case, the system's Oncology will be deployed across the conventional OSI Layer

made up of the; Sensor (Data Collection Layer), Edge Nodes (Edge Layer), Edge Gateway Layer, and AI Processing Layer, (optionally Cloud Layer) if it was a Distributed Edge System.

Table 8

Author/Year	Approach	Impact	Research/Source
Miskovic et al., 2021, "Low Power AI Hardware"	Use low-power processors like ARM-based systems (e.g., Raspberry Pi) or custom ASICs.	Reduces energy consumption in processing.	Miskovic et al., 2021
Jouppi et al., 2017, "Energy Efficient AI Hardware"	Use specialized hardware accelerators for AI tasks, such as TPUs, which are more energy efficient.	Increases performance per watt.	Jouppi et al., 2017
Molchanov et al., 2016, "Model Optimization for AI"	Use model pruning, quantization, and knowledge distillation to reduce model size.	Reduces memory usage and computation cost.	Molchanov et al., 2016
Abadi et al., 2016, "Distributed AI"	Offload AI workloads to edge devices or distribute tasks across multiple low-power devices.	Reduces data transfer energy and optimizes local resources.	Abadi et al., 2016
EAI, 2021, "Sustainable Energy in AI"	Use renewable energy sources to power AI infrastructure (e.g., solar, wind).	Reduces carbon footprint of AI operations.	EAI, 2021
Strubell et al., 2019, "Green AI Algorithms"	Develop AI algorithms designed for lower energy consumption by focusing on efficiency in both hardware and software.	Decreases overall computational energy.	Strubell et al., 2019
Hall et al., 2021, "Carbon-Aware AI Development"	Implement algorithms that optimize based on environmental impact, adjusting processing based on real-time power grid emissions.	Reduces operational carbon footprint.	Hall et al., 2021
Silva et al., 2021, "Cloud Platforms with Green Initiatives"	Leverage cloud services powered by renewable energy, such as AWS, Google Cloud, or Microsoft Azure.	Reduces individual energy needs for AI systems.	Silva et al., 2021
Wu et al., 2020, "Eco-Friendly Data Centers"	Design data centres using energy-efficient cooling and sustainable building practices.	Minimizes the environmental impact of AI processing infrastructure.	Wu et al., 2020

The Raspberry Pi Model 5B, will be using the Small-scale CNN, ML models for embedded applications, in particularly we will be using Conv1D model in order to process the data. In addition to this, we will used a 1 -Dimensional Convolutional Neural Network in order to process Time Series Data, as opposed to Thermal Images (which typically uses a 2 – Dimensional approach) for conducting the classification tasks as it was no longer feasible to develop, and properly test the performance of a Convolutional 2 – Dimensional Neural Network (Conv2D). This is no implemented as Future Research (work) since the MLX90640 sensors did not collect thermal images (data collection) consistently – presenting inaccuracies and unreliable data (extremely unclean). If this was the case, there could have been

interpolation to predict the missing data values – however this would only be a strategy to compensate for the hardware failure/outliers of the MLX90640 Thermal Array Imaging camera.

Upon research, there was a common pattern of specific sensors/equipment used in order to carry out the experiments and carry out the methodology in regards to developing and acquiring the hardware for Occupancy Monitoring Projects/Systems.

Table 9

Platform	Type	Suitable AI Models	Hardware Characteristics	Use Cases	Source
FPGA (e.g., Xilinx, Intel)	Programmable Logic	CNN, RNN/GRU, Custom Accelerators for AI	Highly parallel, customizable hardware; low-latency, low power consumption	Real-time processing, hardware acceleration	**Givargis et al., 2020; Zhang et al., 2021**
ASIC (e.g., Google TPU)	Custom Chip	Transformers (BERT, GPT), CNN	Task-specific chips for high throughput and energy efficiency	Large-scale AI workloads, Data centres, AI research	**Vaswani et al., 2017; Rasmussen et al., 2021**
SoC (e.g., Qualcomm	System on Chip	CNN, RNN/GRU, LSTM, small models for edge AI	Integrates CPU, GPU, and often AI accelerators; versatile but lower power	Edge AI, IoT, robotics, smart devices	**Cheng et al., 2019; Yao et al., 2020**
Raspberry Pi	Single-board Computer	Small-scale CNN, ML models for embedded applications	ARM-based CPU, limited GPU, low power consumption	Educational tools, robotics, simple ML tasks	**Franke et al., 2021; Zhang et al., 2019**
STM32	Microcontroller	Simple ML models (e.g., TinyML, decision trees)	Low-power ARM Cortex-M cores, hardware accelerators for specific tasks	Embedded systems, IoT, real-time sensor processing	**Torres et al., 2020; Chen et al., 2018**
ESP32	Microcontroller	TinyML, simple CNN, RNN/GRU	Dual-core, Wi-Fi, Bluetooth; low power consumption; good for edge AI	IoT, sensor data processing, smart devices	**Lai et al., 2020; Liu et al., 2021**
PIC (e.g., Microchip)	Microcontroller	Small decision trees, basic ML models	Simple, low power, less memory, real-time processing capabilities	Low-power, real-time embedded applications	**Zhou et al., 2020; Yang et al., 2019**
Arduino Uno	Microcontroller	Basic ML models (e.g., decision trees, linear regression)	Limited memory and processing power; used in simple tasks and prototyping	Educational use, prototypes, simple IoT projects	**Jiang et al., 2019; Guo et al., 2020**
Nvidia Jetson Nano	Embedded System	CNN, RNN, Transformers, reinforcement learning	GPU-powered, ARM-based CPU, powerful for edge AI, high power consumption	Autonomous systems, AI-based robotics, IoT	**Khan et al., 2020; Zhang et al., 2020**

According to the work of (Reference) Evaluation Techniques used to measure performance, and various other important critical factors in determining the quality of the trained model are as follows:

- Accuracy
- Precision
- Recall

- F1 Score
- ROC-AUC
- Log Loss
- Confusion Matrix
- Classification Report

Table 10

Author/Year	Approach	Scenario	Explanation	Source(s)
Strubell et al. (2019)	Energy Consumption of Models	Training large models on GPUs, TPUs, etc.	Energy consumption is determined based on power usage of hardware and training time. Multiply power (W) by time (h) to get kWh.	Strubell et al., 2019: *Energy and Policy Considerations for Deep Learning in NLP*
Lacoste et al. (2020)	Carbon Emissions	Models trained in data centres or clouds	Carbon emissions are calculated by multiplying energy consumption (kWh) by the carbon intensity of the energy source (gCO2/kWh).	Lacoste et al., 2020: *The Carbon Footprint of Machine Learning Training in the Cloud*
Lacoste et al. (2020)	Energy Efficiency Considerations	Power-efficient hardware or algorithms	Training with hardware that uses less power, or optimized models can reduce total carbon emissions.	Lacoste et al., 2020: *The Carbon Footprint of Machine Learning Training in the Cloud*
Strubell et al. (2019)	Reduction of Carbon Footprint	Model pruning, quantization, and distillation	Techniques like model pruning or distillation reduce model size and complexity, leading to lower energy consumption and emissions.	Strubell et al., 2019: *Energy and Policy Considerations for Deep Learning in NLP*
Li et al. (2018)	Optimization and Model Size	Using smaller or more efficient models	Smaller models, such as Efficient Net, require fewer resources, reducing overall emissions.	Li et al., 2018: *Efficient and Interpretable Deep Neural Networks for Edge Devices*

Finally, to conclude the literature review, it has become evident that the best transformation techniques suited towards artificial intelligence on the edge, would be the PCA (Principal Component Analysis) and the Time Series Analysis (TSA) according to the experimental design and methodologies explored by ()

IV. DESIGN

A. System Requirements

The system requirements, of the Raspberry Pi would be to made up of cost-effective electronic components, the encapsulation compared to the industry-standard Smart Sensors which can range from $150 onwards, this will also create a more environmentally friendly approach towards monitoring, and less carbon emissions required.

Additionally, the system will feature a user-friendly interface to display real-time environmental data, such as temperature and humidity levels, along with alerts that activate when conditions deviate from established thresholds. This interface will be developed inside a website, which will behave as a web application initially as a prototype before the final design and implementation on the application on the smartphone. Additionally, to supplement the monitoring of the temperature/humidity measurements from the sensors, we will be using the same dataset to calculate occupancy levels. The **Occupancy** level is measured with a high accuracy rate, ideally over 95%, to guarantee the output will be accurate, and validated for research purposes, and future work. In addition to this, the validation process will be across different processing devices, e.g. **STM32** and the **Raspberry Pi (Pico)** measuring the performance and balancing the cost-effectiveness of the systems.

Designed for on-device computation (REFERENCE), the project avoids a distributed edge or cloud-based setup. (REFERENCE) This means all data processing and decision-making will occur directly on the device without task offloading, enhancing privacy and reducing latency. (REFERENCE)

Each device has 2GB of RAM, with 300 MB to 570 MB typically available for additional tasks, and an ARM Quad-core Cortex-A53 CPU with around 20% to 30% available compute power. (REFERENCE)

The design of the GUI Interface is designed using storyboard and the front-end programming languages, specifically HTML5, CSS, and JavaScript to adequately display/animate the visual alerts. alerting the relevant staff handling the HVAC or Ventilation systems such as maintenance, or technicians of what approach to take supplementary to the type of alert, and clear visual representation displaying the respective temperature, and variable being affect for instance, “Fast Rising Temperature in Human Resources MCST105 – potentially due to High Occupancy”, etc.

An intuitive control panel (Graphical User Interface) will allow users to interact with the system, while simulated automated responses can be recommended based on forecasted trends, or to

provide an accurate depiction of how the environmental variables such as hotspots for high humidity amounts which can contribute to the production of moss, and fungi infection.

The purpose of training the ML model, will be used to forecast upcoming health hazards, specifically within extreme ranges indicating high temperature changes in the room/spaces. Hence, if there was detected elevated spikes, the model should be designed, to respond to these anomalies by triggering a response to communicate the findings through (MQTT – Message Queing Telemetry Transfer Protocol/HTTP – Hyper Text Transfer Protocol) protocol to a website, displaying a simple 2D Graphical User Interface.

A. Project Requirements

This project uses the Raspberry Pi Model 5 with Debian Linux (Raspbian Lite OS) as the operating system, chosen for its low power consumption, light-weight operating system (REFERENCE) and open-source accessibility. (REFERENCE)

Using Python libraries such as Pandas and NumPy for pre-processing is essential to ensure the data has gone through the data cleaning stages, and is ready for analysis and training models using them (.csv datasets).

In the process of helping monitor the environmental variables, (in this case. the temperature and humidity) inside the office environment, various strategies will be implemented which will be used when handling the dataset (obtained from the Experiments run and validated). The following datasets tend to be closely related to large-size datasets (Big Data), however in this case we will only be exploring the strategies in respect to compressing the large datasets (ideally from sensor readings).

Contrary to the dataset evaluation from various researchers, and other institutions focusing on implementing these strategies, and not exclusively environmental variables.

In the first instance, we will be exploring the strategy employed by (Bengio et al., 2015), focusing on developing and collecting data through scheduled patterns in a time-frame, specifically "Scheduled Sampling", for forecasting/predicting sequences with Recurrent Neural Networks (RNN), as opposed to my implementation of Convolutional Neural Network (CNN). This would be directly applicable, and a suitable type of minimal dataset strategy in experimental design works – in particularly when collecting raw data from specific rooms in rooms/environments in a building environment. The Strategies used

here, as follows: In our case, since the datasets that were used in order to train the model happen to be very large (over 100MB+), to allow for faster training time of a deployment-ready model.

The Bayes Native Classifier algorithm was used here to reduce the dimensionality of the datasets. The Linear Interpolation algorithm was used to create smoother real-time output feedback on the OLED Display.

The PSA Analysis, to compress the dataset into a dataset that uses valuable features but needs reducation.

Monitor temperature, humidity, air quality (particulate matter), and occupancy using a variety of sensors. Detect unusual or abnormal changes in environmental conditions by capturing data from multiple points within each room. Analyse the collected data to identify patterns, trends, and potential room conditions and usage concerns.

In terms of model development, machine learning-specific algorithms, like linear regression models or convolutional neural network models should be selected as a design choice, since it is most suited to time-series data (sequential data). Due to the nature of the available dataset, the final decision to choose the Convolutional Neural Network Architecture (specifically Conv1D, and Conv2D), can be justified since it is most suited towards handling a wider variety of sensor input (data). Hence, to be trained to perform more complex tasks, (in this case) perform data prediction and classification tasks. (Guidici and Clark, 2017; Kattenborn et al., 2021)

Edge computing devices will be crucial for running models locally, reducing latency, and ensuring real-time data processing. Frameworks such as TensorFlow Lite can help deploy models on edge devices. (Jouini *et al.*, 2024)

Lastly, the system needs to be optimized for energy efficiency and scalability to handle large office spaces. Continuous monitoring and testing within real environments will be key to refining the model's performance and ensuring low latency (real-time performance) using low-power microcontroller/electronic systems (typically under 8V supply voltage).

Table 11

Author/Year	Encapsulated Case	**Materials**	**Deployment**

Henar Mike O. Canilang Wansu Lim , Angela C. Caliwag , James Rigor C. Camacho, , Senior Member, IEEE, and Martin Maier		The Materials suggested are plastic, or	The CNN MCTS system is deployed on the NVIDIA Jetson, and uses the framework
Geir Kulia CEO, Trude Marit Risnes Chair of the board, Lars Førland Havstad CTO, Tommy Nordbøe CFO, Magne Dåstøl Business developer, Jon Tingvold Senior software engineer, Øystein Midttun Senior Researcher)			Deployment on unknown system, unknown hardware requirements
László Balogh Chief Technology Officer			Deployment on FLIR Lepton Infrared Sensor (using fisheye camera)
DINESH WADHWANI CEO FOUNDER DANNY WADHWANI CFO			

Assuming, the project will be run using a Raspberry Pi as the main MCU which runs the environmental monitoring project using all the cores accessible to it, (in this case) Quad Core ARM Cortex. This can create fluctuations, in the delivery and input electronic signals detected by the sensor system, collecting the environmental variable data if there isn't sufficient ventilation or airflow system implemented in the encapsulation case specifically.

Due to this, I have opted to view this as a priority when designing the Encapsulation case on SOLIDWORKs and have opted to add several holes exposing the sensors to the air.

Due to the nature of the temperature fluctuations of the Raspberry Pi, I have opted to use a default Heatsink component that is installed don the Raspberry Pi using several thermal pastes on the processors to allow a cooling mechanism (non-electronic) in the case of high temperatures.

Another approach discussed upon my research, suggests that using a wireless sensor network, e.g. using Z Zigbee or other forms of networking with hardware sensors that are built for monitoring temperature/humidity and other environmental variables would be more intuitive approach specifically when data collection, needs to serve a higher accuracy/quality in comparison to higher throughput.

By expanding the sensor network such that hazards, and risks in relation to the level of electrical current and potential trips/accident due to the long length of the cables connecting the Raspberry Pi to the other layers could present issues. Although this approach has been implemented in typical British Households, in particularly when installing ethernet internet cables.

Table 12

Author/Year	**Material/Focus**	**Key Findings**	**Relevance to Project**
Marie-Laure Locatelli et al. (2014)	Silicone gels & elastomers for high-temp packaging	Elastomer stable up to 300°C, gel cracked at 250°C	Not relevant – RPi system operates at low temperatures
Kinjo & Ogata (1995)	Epoxy Molding Compounds (EMCs)	High mechanical protection, environmental resistance	Relevant – suitable for structural protection of enclosure

A. A. Hamzah et al. (2019)	Polyimide, Parylene C, Carbon-based epoxy resin	Carbon-based epoxy resin offers superior strength	Relevant – strong encapsulant material for mechanical durability
Meghavin Bhatasana & Amy Marconnet (2021)	Phase Change Materials (PCMs) & ML optimization	PCMs reduce peak temperatures, optimized using ML	Not relevant – thermal buffering unnecessary for low-power RPi system
Hisato Yamaguchi et al. (2013)	Reduced Graphene Oxide thin films	Excellent barrier against moisture & oxygen	Relevant – can improve enclosure sealing and durability

Based on research, the materials of the encapsulation case should be used to reduce latency of data collection from edge computing devices.

This has been a very significant aspect of developing edge computing devices, as it plays a crucial role when determining the effectiveness of the encapsulation cases, and the environmental variables associated with the performance of the processor units, and electronics enclosed inside the encapsulation case.

The selection of encapsulation materials for the Raspberry Pi-based occupancy detection system prioritizes factors such as structural protection, EMI shielding, and environmental resistance, rather than high-temperature tolerance or thermal regulation. Therefore, materials like Epoxy Moulding Compounds (EMC) and carbon-based epoxy resins were considered suitable due to their proven effectiveness in mechanical protection and environmental sealing, as supported by Kinjo & Ogata (2018) and Hamzah et al. (2019). Additionally, reduced graphene oxide thin films (Yamaguchi et al., 2013) could be utilized as an ultra-barrier layer to minimize moisture ingress and enhance long-term reliability. High-temperature soft encapsulants and phase change materials discussed in other studies were not considered relevant, as the operational conditions of the Raspberry Pi system do not require extreme thermal management.

Upon research, I discovered that development boards working with light-weight AI models, have been designed in order to adapt to a range of use case applications, and environments either being software, or hardware based. Several examples of industry-grade ready to deploy board are for instance, **Sipeed MAIX GO, NVIDIA Jetson Nano, Google Coral Dev Board, Arduino Nano 33 BLE Sense, Raspberry Pi Zero 2 W.**

Table 13

Development Board	Processor	AI Capabilities	Dimensions	Reference
Sipeed MAIX GO	Kendryte K210 dual-core RISC-V	Supports TensorFlow, Tiny-Yolo, MobileNet V1; includes FPU and KPU for AI acceleration	Compact design	Sipeed MAIX : The World First RISC-V 64 AI Module
NVIDIA Jetson Nano	Quad-core ARM Cortex-A57 CPU; 128-core Maxwell GPU	Supports frameworks like TensorFlow and PyTorch; suitable for AI applications	100mm x 80mm	NVIDIA Jetson Nano Developer Kit
Google Coral Dev Board	Quad-core Cortex-A53 CPU; Edge TPU coprocessor	Capable of 4 trillion operations per second; optimized for TensorFlow Lite models	88mm x 60mm	Dev Board Datasheet
Arduino Nano 33 BLE Sense	ARM Cortex-M4 CPU	Supports TensorFlow Lite Micro; suitable for simple AI tasks like voice recognition	45mm x 18mm	Nano 33 BLE Sense
Raspberry Pi Zero 2 W	Quad-core ARM Cortex-A53 CPU	Supports lightweight AI models; compatible with various AI frameworks	65mm x 30mm	Raspberry Pi Zero 2

Table 14

Osman et al., 2021	Lightweight	TinyML on microcontrollers	Emphasizes the importance of lightweight frameworks like TensorFlow Lite Micro for resource-	TinyML Platforms Benchmarking
David et al., 2020	Lightweight	Embedded systems	Introduces TensorFlow Lite Micro, designed for efficient execution on embedded systems with	TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems
TechRepublic, 2002	Comparison	Software development methodologies	Discusses the advantages of lightweight methodologies being	Heavyweight vs. Lightweight Methodologies: Key Strategies for

The Particle Matter Sensor, is typically used when measuring real-time Occupancy/Vanacy as it is typically used in various research papers on the similar domain of Occupancy Monitoring according to the works of (Cretu, Stamatescu and Stamatescu, 2021; Acquaah *et al.*, 2023; Anderson and Ehrett, 2023)

Table 15

Author(s)	**Year**	**Title**	**Source/Reference**
Meng, L., Li, Y., Du, J., et al.	2023	Task Offloading Optimization Mechanism Based on Deep Neural Network in Edge-Cloud Environment	*Journal of Cloud Computing*, 12(1). https://doi.org/10.1186/s13677-023-00450-6
Qin, Y., Li, X., Song, Y., et al.	2025	Task Offloading Optimization in Mobile Edge Computing Based on a Deep Reinforcement Learning Algorithm Using Density Clustering and Ensemble Learning	*Scientific Reports*, 15, Article 84038. https://doi.org/10.1038/s41598-024-84038-3
Chen, L., Zhao, D., Sun, Q., et al.	2022	Romanus: Robust Task Offloading in Modular Multi-Sensor Autonomous Driving Systems	arXiv preprint. https://arxiv.org/abs/2207.08865
Yan, J., Zhao, F., Wang, W., et al.	2018	Optimal Task Offloading and Resource Allocation in Mobile-Edge Computing with Inter-user Task Dependency	arXiv preprint. https://arxiv.org/abs/1810.11199

Mao, Y., Zhang, J., Letaief, K. B.	2016	Dynamic Computation Offloading for Mobile-Edge Computing with Energy Harvesting Devices	IEEE Journal on Selected Areas in Communications, 34(12), 3590–3605. https://arxiv.org/abs/1605.05488

I have opted to use the **Tiny Linux**, as an alternative in comparison to the **Raspbian Lite** or the **Circuit Python** which is typically recommended for real-time monitoring. This is due to the nature of the minimalism and low GUI configuration which can affect the latency and background operations inhibiting the real-time ability.

Upon observation, I discovered that using FreeRTOS or other minimal operating systems are also effective in particularly when trying to interface other sensors/equipment in particularly when the applications are based on objects that are actively monitoring (non-static), or low-power requirements, e.g. Drones.

The easiest framework/programming language to interface with edge network would actually be python, since it has a less vast knowledge curve in particularly when considering the time scale expected in order to learn the functions such as print("'"), and while loops, i.e. while().

Tensorflow Lite, Tensorflow lite for microcontrollers, ONNX, and Edge impulse SDK are the names of the few various programming frameworks found exclusively in Python. Although, based on the information from the references on, it displays clearly the throughput levels and the efficiency of running Machine Learning models on

We can measure which framework consumes the less/most electricity/power consumption based on the monitoring tools provided by third-party software in the Linux environment. We can also measure the framework which requires less resources to run, by also monitoring the level of RAM that is being used when the models are being run, similarly how the RAM is monitored using Task Manager.

Table 16

Framework Library	Development Language	Edge Device Requirements	Open Source	Task	Applications
TensorFlow Lite	C++, Java	Mobile Embedded Device	Yes	Inference	Computer Vision, Object Detection
Caffe2	C++	Multiple Platforms	Yes	Training and Inference	Image Analysis, Video Analysis

Core ML	Python	Apple Devices	No	Inference	Image analysis
MXNet	Python, C++	Multiple Platforms	Yes	Training and Inference	Image Recognition, Text Recognition
PyTorch	Python	Multiple Platforms	Yes	Training and Inference	Image Recognition, Text Recognition
AWS IoT Greengrass	Python, Node.JS, Java, C, C++	Multiple Platforms	Yes	Inference	Precision Agriculture, Autonomous Vehicles
Edge2Train	C++	MCUs supported by Arduino IDE	Yes	Training and Inference	Video Analysis
OpenAI	â€“	Multiple Platforms	Yes	Training and Inference	Various Applications
TensorRT	C++	NVIDIA GPU	No	Inference	Image Classification
Deep Things	C/C++	Multiple Platforms	Yes	Training and Inference	Object Detection

A Significant research gap which has been solved is the design and implementation of lightweight AI models, obtained through converting.

Another research gap which has been solved, is the knowledge and applications of using lighter weight Operating Systems, such as Raspberry OS Lite (Raspbian Lite).

Similarly, I also discovered that when interfacing hardware such as the Raspberry Pi 5, you don't need to be restricted to using an operating system – but you are able to approach the hardware using "bare-metal" programming e.g. with Embedded C, or Assembly in order to achieve real-time results, reduce latency.

Table 17

Aspect	**Java RMI (RPC)**	**CORBA**	**REST API**	**WebSocket**
Definition	Allows Java programs to call methods on remote objects.	Language-neutral standard for distributed object communication.	Web-based architecture for client-server communication.	Full-duplex communication over a single TCP connection.
Complexity	Moderate; requires Java setup, RMI registry, and method stubs.	High; requires CORBA IDL (Interface Definition Language) and ORB setup.	Low; straightforward with HTTP methods like GET/POST.	Moderate; requires WebSocket libraries for both ends.
Protocol	Java RMI Protocol (JRMP).	Internet Inter-ORB Protocol (IIOP).	HTTP or HTTPS.	WebSocket protocol over TCP.

Language Support	Java only.	Multi-language support (Java, C++, Python, etc.).	Multi-language support (via HTTP libraries).	Multi-language support (via WebSocket libraries).
Ease of Setup	Requires RMI registry and client-server setup in Java.	Requires CORBA libraries, ORBs, and configuration.	Very easy; no additional setup beyond a web server.	Slightly harder than REST; WebSocket library needed.
Performance	High (optimized for Java-to-Java communication).	High; designed for distributed systems.	Moderate; limited by HTTP request/response cycle.	High; real-time communication with low latency.
Scalability	Limited to Java environments.	Scalable, but setup complexity can hinder small projects.	Highly scalable (stateless design).	Scalable for real-time applications but requires stateful handling.
Real-Time Capability	Moderate; synchronous calls can introduce delays.	Moderate; similar to RMI but depends on implementation.	Low; designed for request/response rather than real-time.	High; designed for real-time, bidirectional communication.
Security	Secure in Java environment; TLS can be added.	Requires custom security configurations.	Built-in support for HTTPS.	Needs additional security setup (e.g., WSS for encryption).
Use Case Fit	Good for Java-only ecosystems.	Best for large, heterogeneous distributed systems.	Best for simple, client-server interactions (e.g., dashboards).	Best for real-time data streaming and updates.

I can use Architecture Blocks in my design since it uses inputs and output, respectively alongside other types of

Table 18

Author / Year	Sensor(s) Used	Architecture Block Used	Application Summary
Riaz et al. (2021)	DHT22, MQ135	Input: Sensor → MCU → Output: Alert LED	Designed indoor air quality and occupancy monitor using gas and humidity sensors
Ahmed et al. (2020)	PM2.5, MQ135, IR sensors	Sensor Layer → Processing Layer → Output	Multi-sensor-based occupancy detection for HVAC energy control

Patel & Shah (2019)	DHT22, PIR, MQ135	Block-level flow from sensors to ESP32	Smart room occupancy detection with environmental context
Rana et al. (2023)	DHT11, PM2.5, CO_2 sensors	Layered Architecture: Sensing–Edge–UI	Edge AI-based real-time monitoring of air quality and occupancy detection
Wu et al. (2017)	IR camera, Gas Sensor (MQ series)	Block Diagrams with Microcontroller I/O	Energy-efficient building occupancy sensing using gas variation and thermal signatures

- **MLX90640** - Thermal Camera Sensor, which utilizes infrared technology and the lens to capture image/dates at a distance/ratio of (INSERT HERE). It utilizes this technology in order to detect different temperatures, as long as one object moves at a time.

- **DHT11** - Temperature and Humidity Sensor

  The setup includes **DHT20 sensors for temperature and humidity monitoring**, and low-cost infrared sensor cameras placed throughout rooms to gather environmental data. Any additional hardware is restricted to components that significantly improve performance while maintaining cost-effectiveness; otherwise, the system will be optimized to avoid adding extra memory or hardware.

- **FC-22 (MQ135)** - The FC-22 (MQ135) Gas sensor was chosen specifically, in order to measure CO levels, and alcohol in the atmosphere.

- **Raspberry Pi 5** - Microprocessor Platform

  The Raspberry Pi 5 Model A is the most updated, and newly released single board computer from Raspberry developed to complete tasks previously run on Raspberry Pi 3 (i.e. older models), with less latency and more efficiency. It has a main processor, which is designed and manufactured by the ARM processor company, specifically Quad Core ARM Processor.

- **Power Supply Module** - To provide stable 5V/3.3V for the system, relying on a supply of voltage (electricity), preferably a portable power bank or re-chargeable lithium batteries.
- **Encapsulation Case (3D – Printed)** - The encapsulation case, which will have a role to be a housing unit for the Raspberry Pi, and the various components used in the Environmental Control Project.

The Programming Environment will more likely use the Thonny IDE environment, with models deployed via **TensorFlow Lite** to ensure they are optimized for the Raspberry Pi's resource constraints. Although up on research, I have also discovered that using Geary Launcher allows me the ability to view how much memory is being used in the respective registers/memory of the **Raspberry Pi**, and monitor the Power Consumption of the ARM Cortex Processor in order to measure the success rate in reducing the level of electricity being used, and the effectiveness of the cooling system.

Table 19

Project requirements	**AVNET i.MX 8M (AES-MC-SBC-IMX8M-G)**	**NVIDIA Jetson Nano**	**Raspberry Pi**
Processing	Quad-core ARM Cortex-A53 + Cortex-M4	Quad-core ARM Cortex-A57 + GPU	Quad-core ARM Cortex-A57
AI and ML	CPU-based, supports lightweight models	GPU-accelerated, efficient for AI	CPU-accelerated efficient for general purpose processing, and non-intensive tasks
Power Consumption	Lower, more energy-efficient	Higher, but manageable with power modes	Higher, but manageable with different operating systems.
Ease of Development	Good, supports Linux	Excellent, supports CUDA, TensorRT, etc.	Excellent supports all programming languages, primarily Python.
Connectivity	Versatile, including Wi-Fi/Bluetooth	Versatile, USB, Ethernet, GPIO	Versatile, USB, WiFi, GPIO, Micro-USB. Ethernet
Cost	£150	$160	£60

Relying upon the needs of the Edge Infrastructure to run, in this case specifically the **Raspberry Pi Model 5**.

We will need to deploy and utilize the resources inside the **Raspberry Pi** to deploy the Edge Server, in comparison to high-end data centres made up of computer networks, combined in order to allow for

resource allocation and provide a higher throughput rate in cases where, Big Data is needed to be pre-processed or used for training Machine Learning models.

The system will not use the **distributed system architecture** found in Fog Computing, but strictly on device processing which completely removes any need to create any type of linux instances using a distributed system of the cloud and the edge , i.e. Multi-acess Edge Cloud, which is industry-standard for similar type of applications. Hence, this will create a more cost-effective approach since, there is no requirements or subscriptions involved of using cloud services provided by third party companies e.g. AWS, Azure, GCP etc.

The design's hardware implementation was initially designed using the KiCAD software to deploy the hardware components, inc. the resistors/capacitors/switches/timers to allow the interfacing of specific sensor nodes, and adaptors to enable portable power supply (i.e. heavy duty re-chargeable battery/power banks).

The Programming Environment will more likely use the Thonny IDE environment, with models deployed via **TensorFlow Lite** to ensure they are optimized for the Raspberry Pi's resource constraints. Although up on research, I have also discovered that using Geary Launcher allows me the ability to view how much memory is being used in the respective registers/memory of the **Raspberry Pi**, and monitor the Power Consumption of the ARM Cortex Processor in order to measure the success rate in reducing the level of electricity being used, and the effectiveness of the cooling system.

Relying upon the needs of the Edge Infrastructure to run, in this case specifically the **Raspberry Pi Model 5**.

We will need to deploy and utilize the resources inside the **Raspberry Pi** to deploy the Edge Server, in comparison to high-end data centres made up of computer networks, combined in order to allow for resource allocation and provide a higher throughput rate in cases where, Big Data is needed to be pre-processed or used for training Machine Learning models.

The system will not use the **distributed system architecture** found in Fog Computing, but strictly on device processing which completely removes any need to create any type of Linux instances using a distributed system of the cloud and the edge , i.e. Multi-access Edge Cloud, which is industry-standard for similar type of applications. Hence, this will create a more cost-effective approach since, there is no requirements or subscriptions involved of using cloud services provided by third party companies e.g. AWS, Azure, GCP etc.

The design's hardware implementation was initially designed using the Ki CAD software to deploy the hardware components, including the resistors/capacitors/switches/timers to allow the interfacing of specific sensor nodes, and adaptors to enable portable power supply (i.e. heavy duty re-chargeable battery/power banks).

VI. CHALLENGES AND ETHICAL IMPLICATIONS

A. Data Privacy and Security

The use of AI involves collecting large amounts of data, raising concerns about data privacy and security. Protecting sensitive information is crucial. Algorithmic Bias AI systems can exhibit bias if not trained on diverse datasets. In environmental monitoring, biased algorithms

B. Ethical Considerations

The application of Artificial Intelligence, in air quality/occupancy monitoring raises ethical questions, including the impact on local communities and the prioritization of environment conservation efforts. Establishing ethical guidelines and engaging with stakeholders is vital. (van Wynsberghe, 2021)

On other occasions, where the level of occupancy is measured, the use of sensors within a workplace or classroom full of students can potentially present unethical practices if it is misused for financial gain by selling this data to third party companies that have commercial objectives, without the consent of the participants/subjects being measured.

The consequences of the data becoming leaked, can lead to bias competition and purposes of (Capitol Technology University Blog, 2023).

In this case, the project was designed to meet these considerations of ethical practices since it doesn't allow any Building Manager/Administrator to view the camera feed or specifically, a higher resolution video/image data. By doing so, it protects the identity and anonymises the occupancy motion and movement in the office spaces. Otherwise, in previous building management software systems, like in (REFERENCE to Honeywell) it can compromise the identity of the occupants – by using non-intrusive sensors and low-resolution thermal camera data.

C. Environmental Considerations

The sensor system was designed with low-power hardware (e.g., Raspberry Pi 5, ESP32) to minimize energy consumption. However, sustained operation in varying temperature and humidity environments posed challenges for sensor accuracy and durability. Additional environmental noise (e.g., airflow, light interference) also impacted the consistency of sensor readings. Ensuring real-time performance while keeping thermal and power footprints minimal was a key constraint in model deployment.

D. Technical Limitations

Artificial Intelligence systems, in general relatively face technical challenges in data accuracy, due to the nature of how neural network architectures perform, in different scenarios. In particularly real-time processing requirements for fast results designed for alarm systems, or accuracy obtained for medical applications, or fault detection.

Based on the hardware components, accessibility for the Environmental Monitoring system didn't present any limitations, since there was a variety of Distributors willing to export, ship, or either obtain scare products from other parts of the World, such as China, e.g. Digi Key Industries. However, since the project requirements had a cost-effective element – which in turn limited the scope of performance, accuracy, and interfacing of the components itself, it presented more engineering problems to solve, e.g. using multiple sensors, i.e. sensor fusion rather than a top-of-the-range Lepton FLIR Sensor which can complete the same task with higher accuracy, and faster processing, etc.

To combat these limitations, we need to continue research and development in regards to the advantages and disadvantages that come with using specific components or configurations. Key engineering challenges observed included power constraints, sensor placement, OS overhead, and limited I/O capabilities, particularly when using low-cost microcontrollers and Raspberry Pi boards. Sensor-specific issues like limited field of view, low resolution, or refresh rate affected real-time data acquisition, which are critical for applications like thermal occupancy detection. Software bottlenecks, such as threading inefficiencies and high dashboard complexity, further demanded minimalistic designs and optimized pipelines. These challenges were mitigated by adopting strategies like multi-sensor fusion, edge model quantization (e.g., TensorFlow Lite), and power management techniques. Inexpensive yet reliable materials were used for protective enclosures, with ventilation designs ensuring proper airflow without sacrificing structural integrity. As such, developing robust AI-based monitoring

systems on constrained hardware is achievable but requires a careful balance of performance, cost, and modularity.

VII. IMPLEMENTATION

A. Initial Sensor Integration and Setup

1. Setup of Simulated Environment

Due to the hardware malfunctions, and anomalous electronic interference, the last resort to test the accuracy of the CNN model and to validate if the software ran as intended – we used a program to simulate the environmental data incl. Room spaces, and University spaces. Although, we found that this was not required since there was available open-source datasets for environmental monitoring and occupancy which achieved similar/better results since there was no discrepancy in regards to hardware failures or overheating.

2. Setup of Headless Raspberry Pi

Since using a full operating system (Raspbian OS) wasn't feasible, and created overhead when e.g. running the program, we resorted to implement the program scripts, alongside interfacing with i2c protocols and testing the program using Raspbian OS Lite. This approach required enabling SSH and pre-configuring Wi-Fi settings via the Raspberry Pi Imager advanced options to connect to the Internet Hotspot, since the University's firewall policies limited the use of SSH ports, enabling remote access (e.g. from Room to RPi location). The operating system was flashed to the MicroSD card using the Pi OS Imager (software), which enabled us to preconfigure settings before installing the operating system.

Once connected, the Raspberry Pi 5 was enabled to perform live data acquisition from connected sensors, and to facilitate real-time classification tasks using pre-trained embedded machine learning models deployed from the Edge Impulse Studio.

The use of the headless setup provided a streamlined workflow for deploying machine learning models without the need for a graphical interface, contributing to the low computational overhead of the system.

Setting up the X2Go Client

I have opted to use the built-in hardware instead of any cloud interfaces, which are built using dedicated hardware, e.g. high-end graphic cards, and not a cost-effective option, in comparison to

utilizing the RPi to host the server, where the Linux instance can be deployed and accessed remotely as long as the same WiFi connection is shared, (in this case, the Mobile Hotspot)

Edge Impulse Studio Connection and Configuration

Upon successful installation of the Edge Impulse Linux CLI, the Raspberry Pi 5 was authenticated and connected to the Edge Impulse Studio platform. This was achieved by initiating the authentication sequence provided by the Edge Impulse website, which involved generating a unique authentication link within the terminal session. This process registered the device within the user's Edge Impulse account and associated the Raspberry Pi with the specific project workspace – that the user used.

API key Integration and SDK Configuration

An API key was generated and configured within the Raspberry Pi environment for advanced model deployment and API-based interaction with Edge Impulse Studio. This key enabled automated data collection, remote management of model deployments, and programmatic access to Edge Impulse services without manual intervention.

The Edge Impulse Python SDK was utilised alongside the Linux CLI, facilitating scripting of various tasks such as data uploading, inference requests, and performance monitoring. This approach ensured an efficient development pipeline while maintaining full control over the machine learning workflow directly from the Raspberry Pi device.

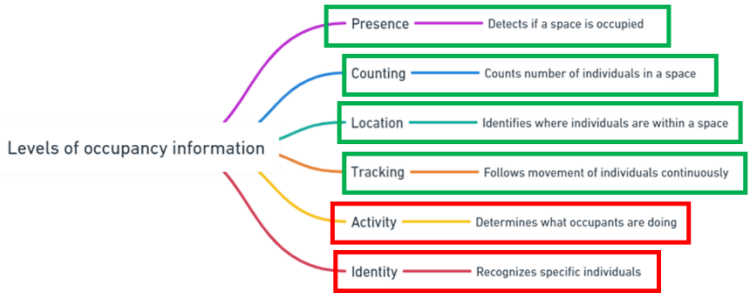
I also attempted to access the SSH session of the Raspberry Pi via my remote computer, however this was not possible, as the Wi-Fi connection provided by the University has disabled connecting on the pot: 22 (SSH) in order to protect their privacy and prevent cybersecurity attacks, so if I wanted to share Wi-Fi connection or connect remotely I resorted to using my hotspot instead. This allowed me to not have a big mess of connecting the keyboard, mouse, and finding space every time to implement/edit the training model and code associated with the Raspberry Pi.

Enhanced Data Augmentation

SDK and Framework

Initially, for prototype purposes and different stages, I opted to use the Edge Impulse SDK in order to collect data faster, and train my models based on pre-built machine learning models (e.g. Conv1D lightweight models, and).

We will be using the TensorFlow framework, as mentioned in the Design section – is more studied towards the specific parameters associated with the development of Edge AI models, and including Model Pruning



Raspberry Pi OS Lite

To reduce the pixelated look and ensure clarity, the edges of each reading were smoothed.

The Raspberry Pi OS Lite operating system will be interfaced using a pre-built OS Imager, which is configured to download and install the Raspberry Pi OS Lite operating system, because the operating system doesn't have any overhead.

I have opted to use the operating system, since it allows me to run Machine Learning models on the Edge without too much interference e.g. desktop environment or operating system (with GUI) as overhead.

Setup of the Convolutional Neural Network (CNN) Model

The model was developed using a Convolutional Neural Network Model, more specifically Conv1D model which only processes "one-dimensional data" - as opposed to "two-dimensional data".

The reason for using this specific model was based on (REFERENCE), where (Author) achieved an accuracy of 99.75% based on public datasets (REFERENCES). I initially wanted to use a Linear Regression, however as the aims & objectives have drifted since the title of the project changed due to the simplicity of running an indoor air quality monitoring system (IAQ) and reduced contribution to

the Engineering field, recreating a system overcomplicated using principles of Machine Learning, - when this is not necessary.

Dependencies and Libraries

For the development and training of the Convolutional Neural Network (CNN) model, the following Python libraries were utilized:

- **Pandas** - For data manipulation and analysis.
- **NumPy** - For numerical operations.
- **Matplotlib & Seaborn** - For data visualization.
- **Scikit-learn** -For data pre-processing, model evaluation, and class balancing.
- **Imbalanced-learn (SMOTE)** -For synthetic minority oversampling.
- **TensorFlow & Keras** -For building, training, and evaluating the Conv1D deep learning model.

I have opted to train, and develop my classification model, based on CNN architecture since it is more suited towards datasets associated with (time series, object identification, and classification tasks).

Although it has also been used extensively, in different architectures (e.g. U-NET CNN) in various separate research papers associated with Occupancy Monitoring using Non-intrusive sensors,

Setup of Data Preparation (Pre-processing)

Data Collection Stage

Using Open Source Datasets (Sensor-specific Output)

The data will be converted into **.hfd5** as opposed to images, for instance ".jpeg" or ".png" formats, and instead extract the temperature values from the pixel array and convert it to numerical values to enable faster processing, in particularly since we are going to process it and act as an input for training the machine learning model.

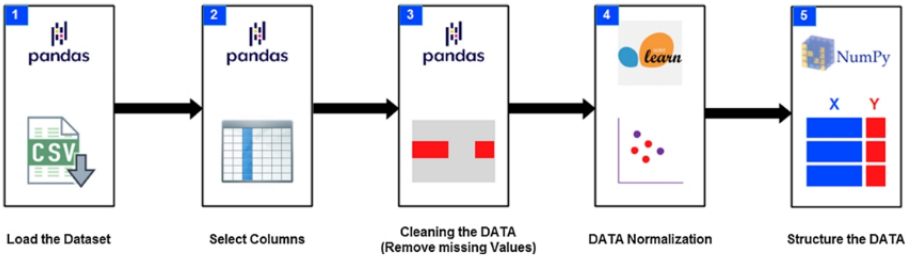


Figure B – Dataset Pipeline

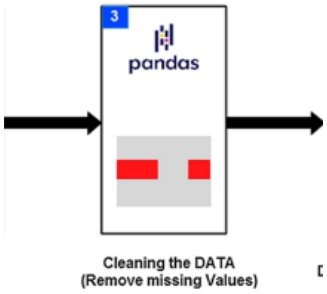
Data Loading and Cleaning

Loading the Data, was very simply implemented since TensorFlow has built-in functionality that assists in uploading/loading a dataset when attempting to train a machine learning model, e.g. CNN in this case. I ensured that the data was .csv, before anything since it was 1-dimensional data formatted in time-series as appropriate.

Typically, in Machine Learning based applications – data cleaning, is an iterative process that consists of several layers which allows us to not only utilize the data using Machine Learning models, but also to ensure that the data from the cells are not only clean, but won't directly/indirectly affect any training parameters e.g. accuracy by providing failures/non-relevant data.

This process consists of the following steps:

- Removal of unnecessary columns such as ts and device.
- Handling of missing values by removing incomplete records.
- Encoding of motion detection as binary (0: No Occupancy, 1: Occupancy).



In the process of cleaning the datasets, we had to initially clean the data so that the additional columns (also known as labels), which would normally be interpreted as features e.g. temperature, match specifically to the training model's expected features.

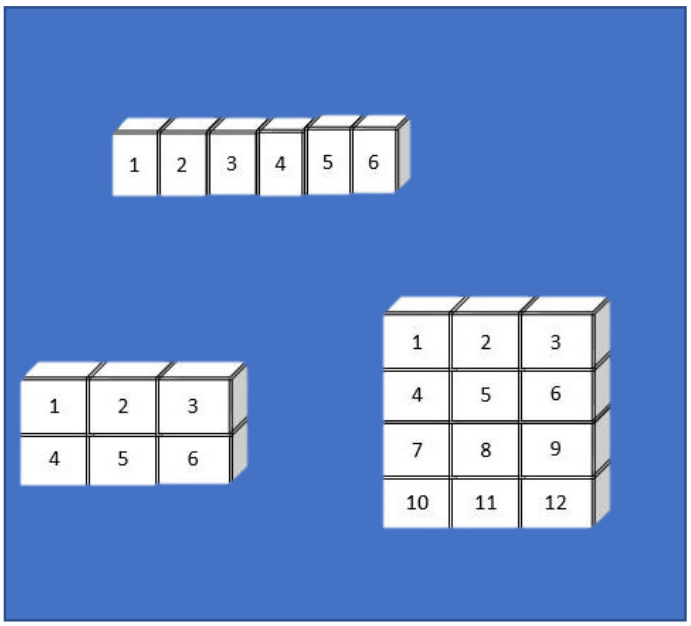


Figure D – 3/2/1 Dimensions of Data

Pre-processing techniques

All sensor features were scaled to zero mean and unit variance using StandardScaler to improve model learning.

Time-series sequences of 100 consecutive sensor readings were created using a **sliding window** technique to enable temporal pattern recognition. In which allowed for the dataset to be processed inside the conv1 model/

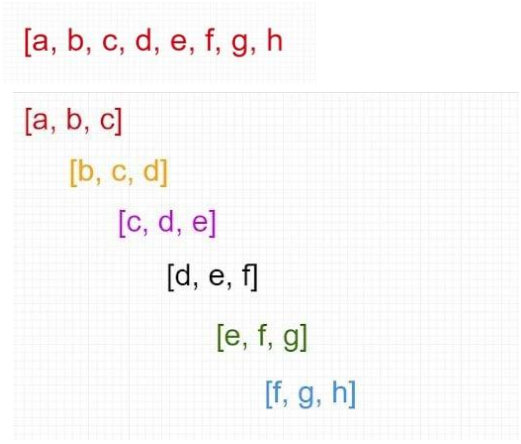


Figure D – Sliding Windows Visual Represetnation using words, e.g. in LLMs to illustrate

The binary motion state at the end of each sequence was used as the target label, indicates the occupancy levels – therefore behaving as a validator for model's classifications of occupancy when gathering evaluation performance metrics when the model is implemented on a validation dataset.

Splitting the Data

The cleaning process involved when attempting to find any anomalies, or values that indicate hardware failure/inaccurate readings was balanced using **SMOTE (Synthetic Minority Over-Sampling Technique)** to address class imbalances

As mention to [reference to dataset], we were able to find, download, and use a range of high quality and well-evaluated datasets, provided by other Institutions or Universities to further assist research in

regards to Occupancy, and Environmental Monitoring specifically. Of the 7 datasets, that was found – only 3 was used it had the largest size/more features matching the desired sensors output to work with.

The first 2 smaller size (.csv) dataset was used for training, as opposed to training/testing purposes. Although, we only applied this training split on to the large dataset (120MB+), specifically 80/20 range.

The largest dataset consisted of approximately 1,800,000 rows of data, made up of 4 combined datasets that came from the same source which resembled each other in regards to the column/features and the origin of the dataset. Although they differed in regards to the location, and of course timescale (timestamp data).

B. Model Optimization for Tiny-ML/Lightweight Models

Model Architecture

A **1D Convolutional Neural Network (Conv1D)** was implemented to capture the temporal dynamics in sensor data.

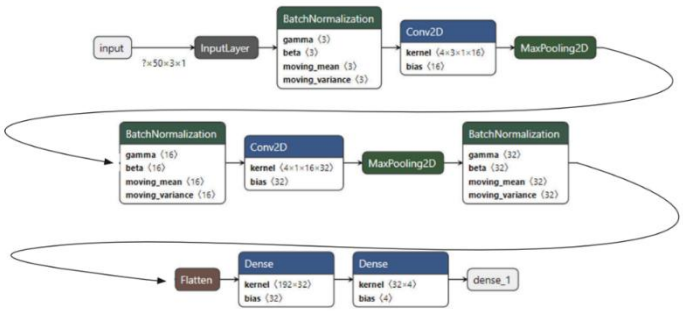


Table 20 – CNN Architecture Layout of using Conv2D Neural Network Architecutre

Table 21

Layer	Parameters
Input Layer	Shape: (100 time steps, 5 features)

Conv1D	64 filters, kernel size = 3, ReLU activation
Batch Normalization	-
MaxPooling1D	Pool size = 2
Conv1D	128 filters, kernel size = 3, ReLU activation
Batch Normalization	-
MaxPooling1D	Pool size = 2
Conv1D	256 filters, kernel size = 3, ReLU activation
Batch Normalization	-
MaxPooling1D	Pool size = 2
Flatten	-
Dense	128 units, ReLU activation
Dropout	50%
Dense	64 units, ReLU activation
Dropout	50%
Output Layer	1 unit, Sigmoid activation (Binary classification)

Batch Normalization

Batch Normalization is another regularization technique used in CNNs, specifically to normalize the output of a (previous activation layer) by subtracting the batch mean and dividing by the batch standard deviation, followed by scaling and shifting. This process helps to stabilize and accelerate training by reducing internal covariate shifts and introduces slight noise in the network for better generalization. (REFERENCE)

The Dropout method was implemented as a regularization technique that is applied to the fully connected layers or convolutional layers in CNNs to prevent "overfitting". (REFERNCE) This is made possible through randomly setting a fraction of activations to zero during the training phrase.
This encourages the network to develop more robust features without relying too much on the weight (values) of specific neurons.
In principle, it prevents the model from Learning undesired habits/activities – since it drops the learning rate (Drop-out), similar to the term "drop out" used when referring to the decision to discontinue Education.(REFERENCE)

Hyperparameter Tuning

The model hyperparameters were tuned empirically based on:

- Filter sizes (64, 128, 256).
- Kernel size = 3.
- Batch size = 32.
- Epochs = 20 with early stopping based on validation loss.
- Learning rate and optimizer (Adam).

Based on this, we were able to carry out testing of the models being trained, with the same hyperparameters and the dataset [69].

The obtained results were: Test accuracy across 5 runs:

- **78.1%**
- **75.2%**
- **74.5%**
- **77.0%**
- **74.1%**

The Mean value obtained from this was, 75.78% Std. deviation: ≈ 1.55%

The fluctuations are low, which is very normal and expected sinc ethe weights of the Neural Network are randomly initialized

That's **relatively stable** — fluctuations are low (within ~±2%), which is **expected** with random weight initialization + early stopping. No huge variance → **your model is reliable**.

Training the Model

The model was trained on the training dataset for 20 epochs. **Early stopping** was applied to prevent overfitting by monitoring validation loss.

The training loss and accuracy were recorded and visualized to evaluate the model's learning performance.

The software that I used to train the model was Lightning-AI, as opposed to the Google Colab which I initially discovered through research was very unreliable when it came to training the model over long term processes, or if the Laptop is in sleep

Make Predictions

After training, predictions were made on the unseen **test dataset**. The model's output was compared against actual labels to assess performance.

Assess Model Performance

The following evaluation metrics were used:

- **Accuracy**
- **Confusion Matrix**
- **Classification Report (Precision, Recall, F1-Score)**

Additionally, training and validation accuracy curves were plotted.

1. Conversion to TFLite Models / TinyML

In order to convert my models into TinyML models, I used the interface of the Edge Impulse, the option is also known as: BYOM (Bring Your Own Model).

This option allows us to not only interface, and upload our own dataset – but also upload our own machine learning model which will be converted into a light-weight models, of our choice e.g. Conv1D for audio classification, which is why I opted to choose tflite instead since it proved to be more appropriate as the program was developed in TensorFlow to begin with.

Deployment

Deployment of Light Weight Model

The trained CNN model was saved in (**H5 format)** for deployment initially as it was the conventional format since it achieved the high accuracy results of (avg. 98%) when it was evaluated using the data splitting principle (80/20).

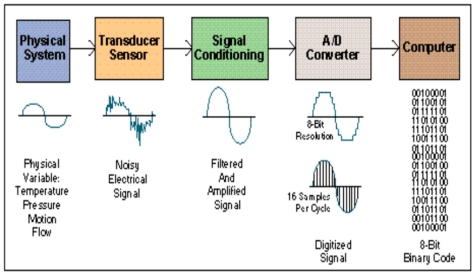
Although I later decided to use TensorFlow API, to convert the model into a TensorFlow lite model (.tflite). This assisted in giving me a realistic evaluation on the performance of the model, especially when it has been quantized and compressed for running on the low-power microcontrollers, (in this case, the Raspberry Pi 5)

Unfortunately, since this was not possible using the Breadboarded Prototype as the cables did not provide a reliable connection (SDA, SCL) – this was counterintuitive to attempt to deploy it as the readings would be have too many anomalies, and provide unrealistic data especially being normalized hence making it "unreliable".

Before recording the results from the deploying the TensorFlow Lite model, I opted to retrain the model using Edge Impulse SDK, to achieve Experimental Results as a comparison to the performance trained by the Edge Impulse SDK and the Conv1D model creation.

Using (Original) Proprietary Datasets (.CSV)

Data Acquisition System



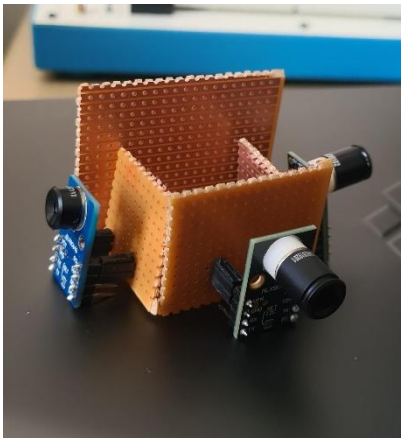
The dataset was created, and the numerical data was exported in a separate .csv files, based on sensor readings collected from multiple environmental-based sensors. This was made possible by with environmental sensors through script-based hardware interaction (on the Python programming language). By utilizing general-purpose I/O (GPIO) and communication protocols e.g. (I2C, SPI, and UART), in respect to each sensors requirements. the paper examines the efficiency of Python-based libraries (e.g., **RPi.GPIO, gpiozero, smbus**) in facilitating data acquisition and control for real-time environmental monitoring. deployed in an indoor office spaces, of which consisted of specifically: **Kitchen, Bedroom, Corridor, Library Group Spaces, Howell Building Office, Conference Room.**

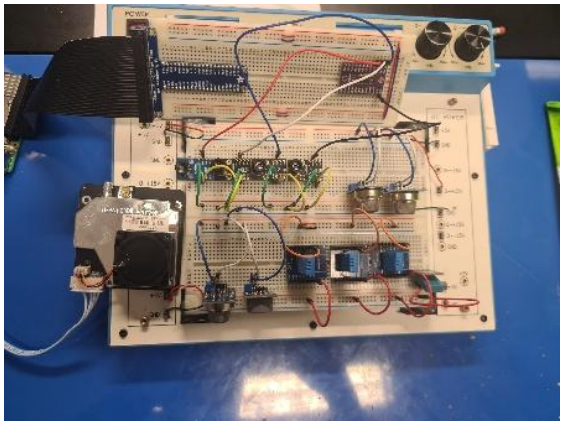
Since, the environmental sensors act as parameter measuring tool, e.g. Temperature/Humidity. For real-time feedback of occupancy from the Environmental sensors, this is required to be generated in real-time, e.g. in samples of 5 secs actively adding rows sequentially.

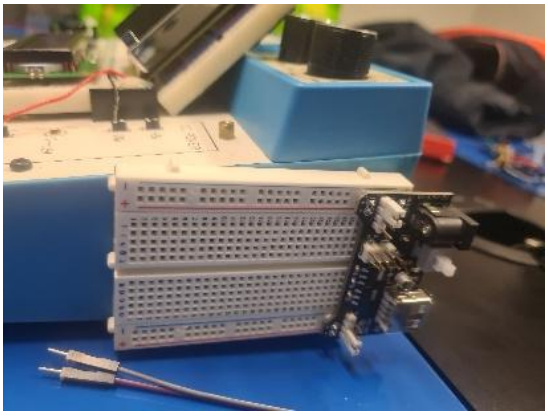
ts	temperature	humidity	pm2_5	motion	CO2 (ppm)	nh33	max_temp
11/04/2025 10:00	23.5	50	25.3	TRUE	co2	0.03	32.5
11/04/2025 10:05	23.7	49	26.1	FALSE	405	0.02	33
11/04/2025 10:10	23.8	48	27.3	TRUE	410	0.03	34.2
11/04/2025 10:15	24	47	25.8	FALSE	395	0.02	32.7
11/04/2025 10:20	23.6	50	24.9	TRUE	420		33.5
11/04/2025 10:25	23.9	48	26.7				
11/04/2025 10:30	24.1						

C. Hardware

Breadboarded Prototype







Implementation of Sensors

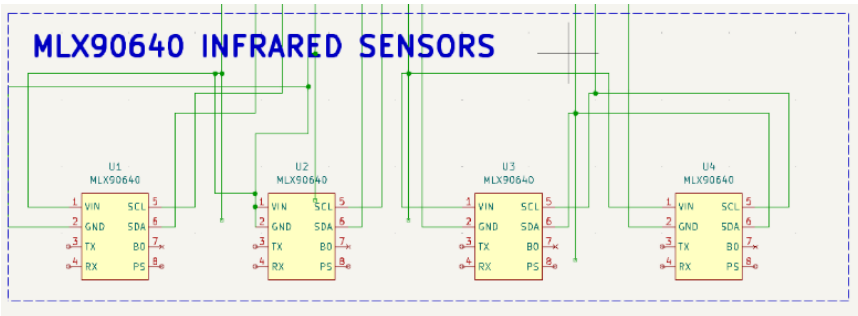
Setup of Thermal Camera

Since, we were required to obtain a realistic output (prototype), to validate the system. I decided to use a breadboard environment, to simulate the system both electronically, and using firmware/software to interface both the hardware (data collection sensors), and the main MCU (Raspberry Pi).

In the Implementation stages, attempting to interface the **MLX90640 Infrared sensor** presented various hardware issues, including the i2c protocol connection wasn't being established, and unread by the i2c detection tool – but this was later corrected by using the raspberry configuration tool to enable the I2C Bus built-in the MCU. In this case, we utilized the Thonny Launcher and the Python that was essential, and used to help me communicate with the embedded sensor using the Python Language (Python version 3.11).

Respectively, the libraries that were used were Adafruit MLX8640, and Blinka– corresponding to the Adafruit manufacturer for the sensors.

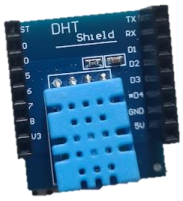
Initially, I opted for obtaining the code for interfacing the MLX90640 from [30], to streamline the process of testing

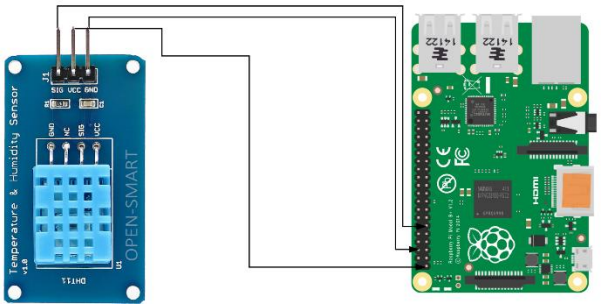


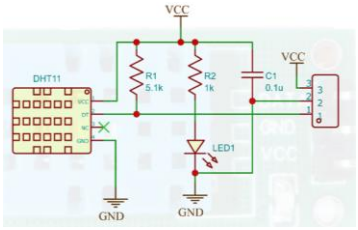
Setup of Temperature Sensors

Obtaining the temperature values, required installing the DHT11 sensors. presented various hardware implementation challenges, as I initially wanted to use the updated DHT20 Temperature/Humidity Sensor but couldn't find the respective libraries to interface and work with the electronic component.

I connected the DHT11 sensor by connecting the respective PIN assignments, (VCC), (GND), (DATA), to the GPIO pins of the Raspberry Pi.

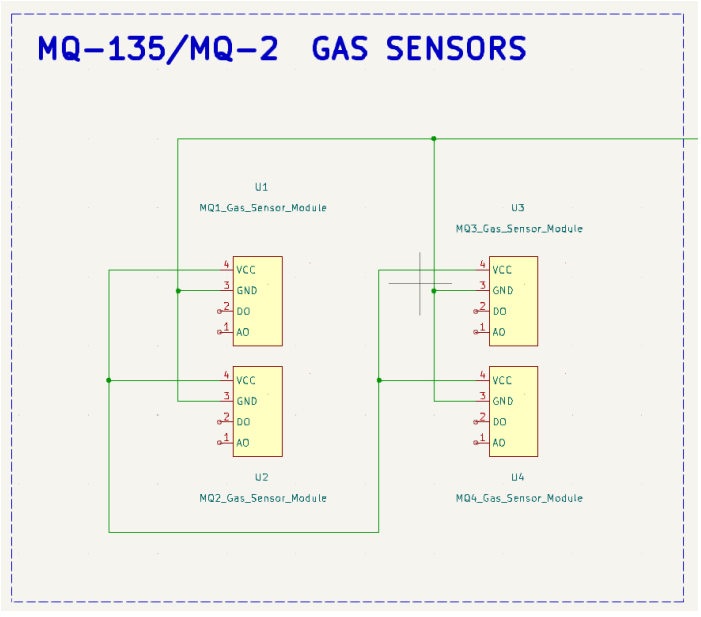






Setup of Carbon Monoxide Gas Sensor

The MQ-135 gas sensor measures air pollutants (e.g. NH_3, CO_2eq, NOx, smoke) and outputs analog VCC was connected to a dedicated 5V power module, which helped ensure a consistent and clean voltage supply. The gas sensor is voltage-sensitive, so a shared power rail on the breadboard was implemented in order to accommodate high-draw components which can cause ripple or dips influencing sensing accuracy/consistency

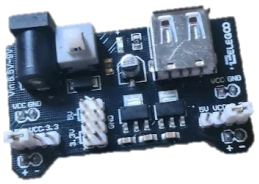


To keep a shared reference level across the circuit, a common GND was sent to the Raspberry Pi, since it will be used to process all of the sensor data and has to deal with all the data pre-processing/post-processing involved. Configured to interact with the Raspberry Pi using SPI protocol, one of the MCP3008 ADC's channels was coupled to AOU (analogue output). The 10-bit resolution of the ADC permitted enough granularity in recording the analogue voltage output of the sensor, therefore indicating the levels of gas concentration.

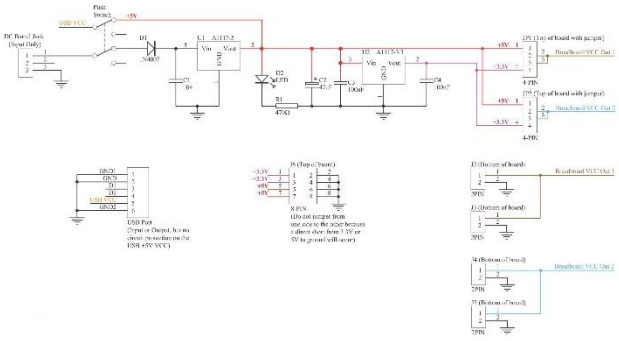
To simulate high gas concentration levels, for instance alcohol spikes. Various controlled tests were carried out using alcohol from perfume and human breath to test out the quality of the sensors. To stabilise its internal chemical components, the MQ-135 additionally needed a warm-up period ranging 24 to 48 hours. Achieving repeatable readings in real-world situations depends on the resistance across its sensing element adjusting to ambient air throughout this period.

Setup of Power Supply Modules

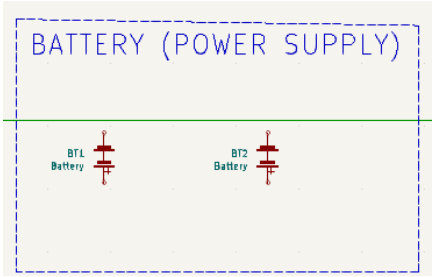
Each sensor group used a dedicated power supply module instead of sharing the Pi's onboard 3.3V/5V lines. This decision was critical for maintaining stability and protecting both the sensors and the Raspberry Pi.

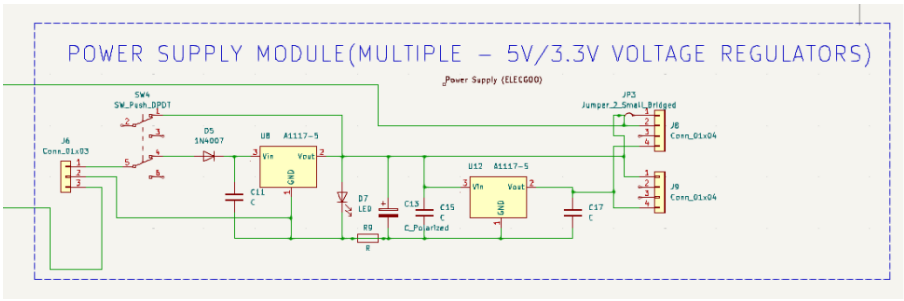


Prevented current starvation: The Raspberry Pi's onboard voltage regulators can only deliver limited current through the GPIO 3.3V and 5V rails. When multiple sensors draw power simultaneously—especially sensors like the MLX90640 with higher peak draw—the Pi may not supply enough current, leading to undervoltage warnings or unexpected reboots. By providing each sensor group with its own regulated power source, each module received consistent voltage and sufficient current without interfering with the Pi's operation.



Reduced the risk of overvoltage to the Pi: External power modules helped isolate power spikes caused by inductive loads or startup surges (like when the MLX90640 initializes). This protected the Pi's internal components and GPIO header from accidental overvoltage damage. It also reduced stress on the Pi's linear regulators, enhancing thermal performance and reliability.





Minimized electrical interference between high-draw components: When components share the same power rail, fluctuations caused by one sensor (e.g. voltage dips during thermal sensor refresh cycles) can interfere with another sensor's accuracy. Isolating the power for each sensor group eliminated crosstalk and power noise, helping maintain signal integrity—especially on analog or I²C lines.

The lithium batteries powering the modules were connected via jumper wires to the center rail of the breadboard. The power modules used were manufactured by ELECGOO, featuring onboard voltage regulators to maintain stable output. While the exact total power draw wasn't calculated, design considerations prioritized current draw and voltage ratings per module.

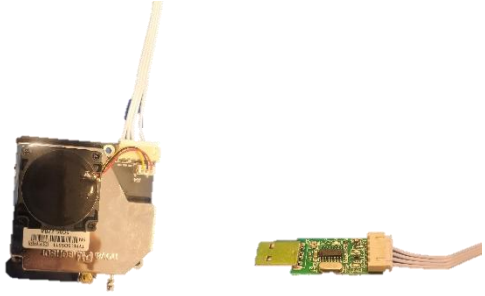
[

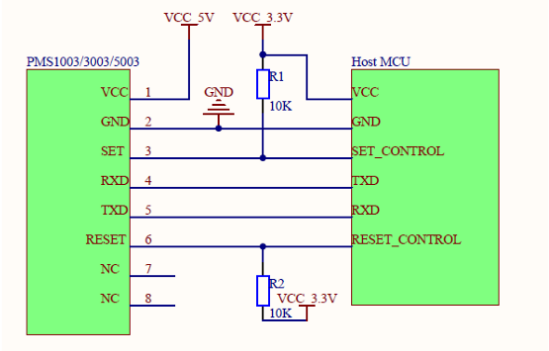
ADC Converter and I2C Multiplexer

MCP3008 ADC: Used to digitize analogue readings from the MQ-135

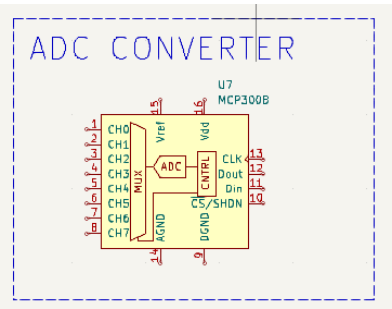
TCA9548A I2C Multiplexer: Allowed multiple I2C devices (e.g., MLX90640) to coexist on the same bus without address conflicts. It also improved signal integrity and allowed controlled I2C communication.

Setup of Particle Matter Sensr

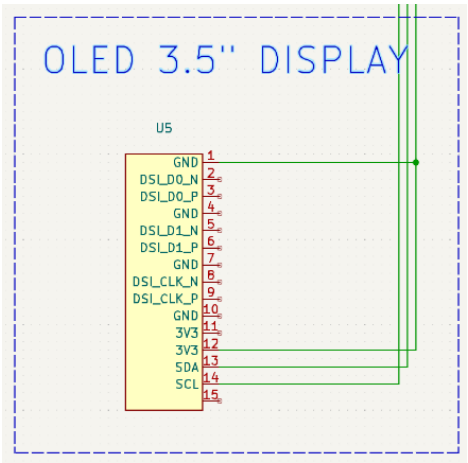




Setup of ADC Converter



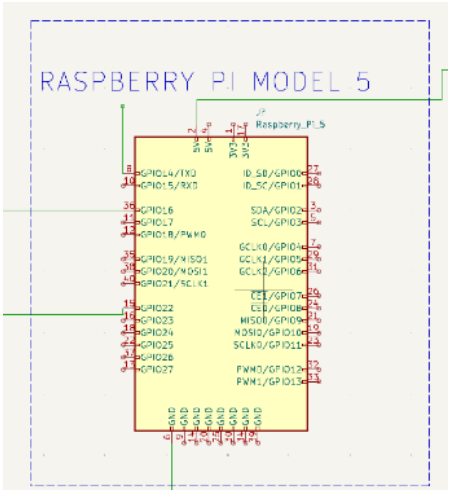
Setup of OLED Display



MCU and SBCs

A. Raspberry Pi 5 Single Board Computer

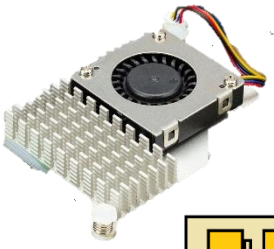
The implementation of the Raspberry Pi 5 Model was very straightforward, since it didn't require any additional modifications to the motherboard, or peripherals e.g. General Purpose Input and Output Pins Although we have opted to use the GPIO extender in order to extend the GPIO pins to the breadboard, as opposed to a manual wiring (soldering)



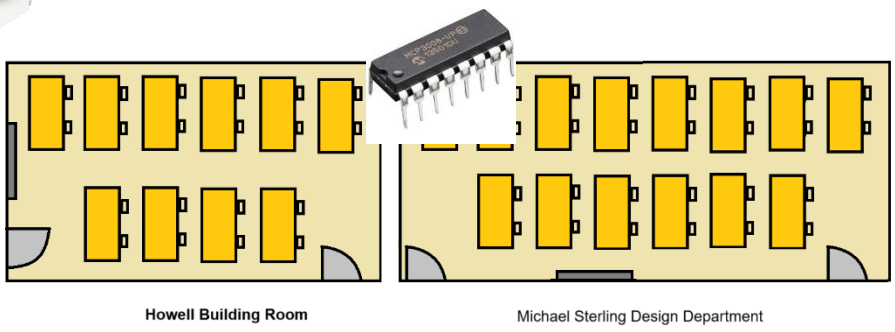
1. HeatSink and Fan

We used the Heatsink, in order to ensure the Raspberry Pi minimalized a health coll temperature as running Machine Leaning Models can strain the CPU – in particularly if the Raspberry Pi doesn't have any optional AI-design constrained processing units, e.g. TPU/AI HAT Processors. (Model reference) Utilizing the Fan, it will allow the temperature to be decreased especially if a model is being run, or in the booting of the operating system (Raspbian Lite OS).

Sensor

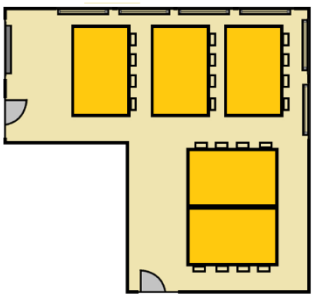


Placement

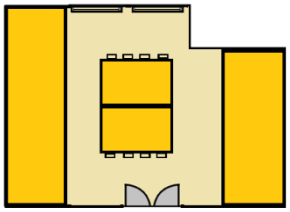


Howell Building Room

Michael Sterling Design Department



Tower C Office Space



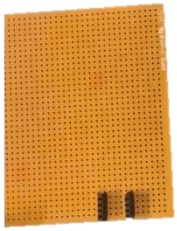
University Hall Kitchen

Deployment

Breadboarded Prototype

Soldering Prototype

We utilized the availability of our University providing soldering equipment, and health and safety equipment to ensure we was following Health and Safety procedures handling dangerous equipment, obtaining high voltage output (high temperature) – soldering pen(REFERENCE to Health and Safety Acquisition Form).

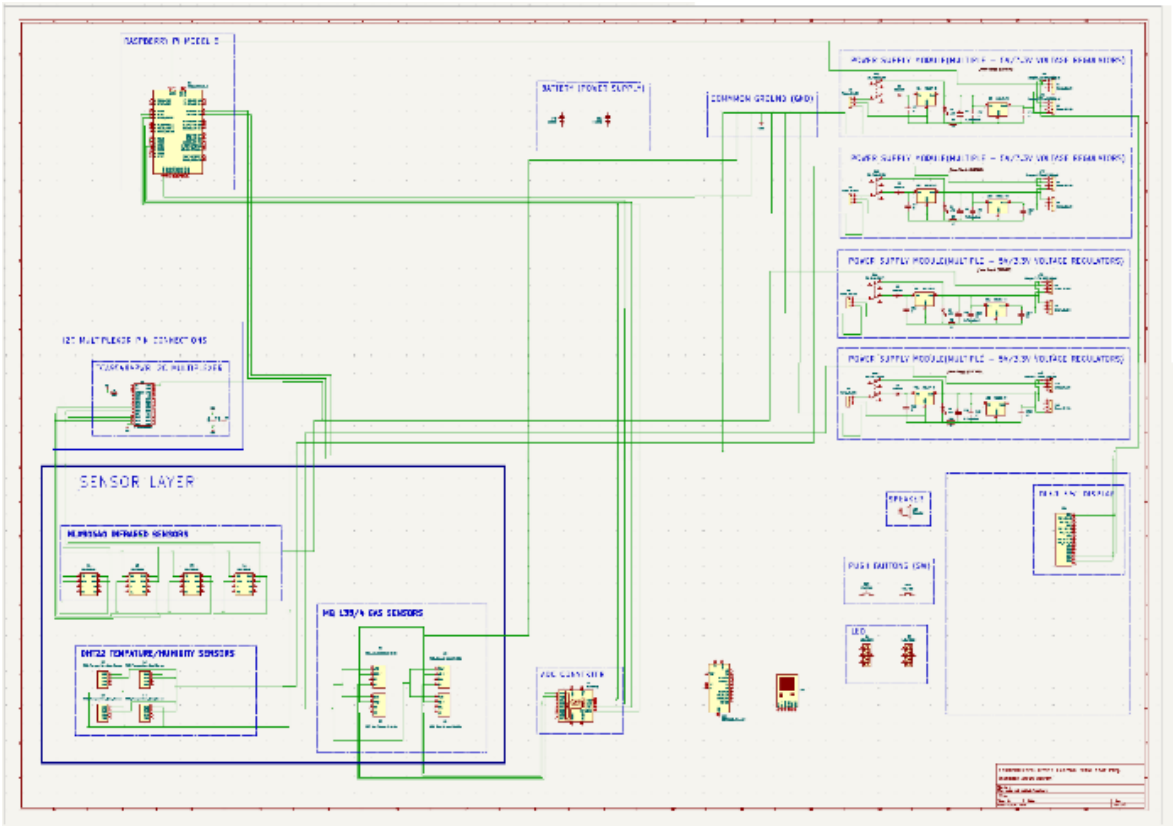


Upon consulting with the Technicians, in the Laboratories we managed to obtain high-quality (brand name) Soldering Prototype boards (mimicking Breadboard) without railing (Figure xx).

We aimed to complete this implementation by the (REFERENCE to Gantts chart date/time).

We referred to the schematic diagram produced by using KiCad software – in order to identify the spaces corresponding to the components (REFERENCE to KiCad).

According to the Schematic, the spaces corresponding to the Raspberry Pi is located on the (left-hand corner) – typically found in MCUs (reference to guidelines about MCU)



VII. METHODOLOGIES

(Results)

Software

Hardware

Ki CAD Schematic Diagram

P-Spice Simulation

Experiment A –

Experimental Analysis of Calibrated vs. Uncalibrated Temperature and Humidity Sensors over different distance lengths and moving/static objects/occupants.

Aim –

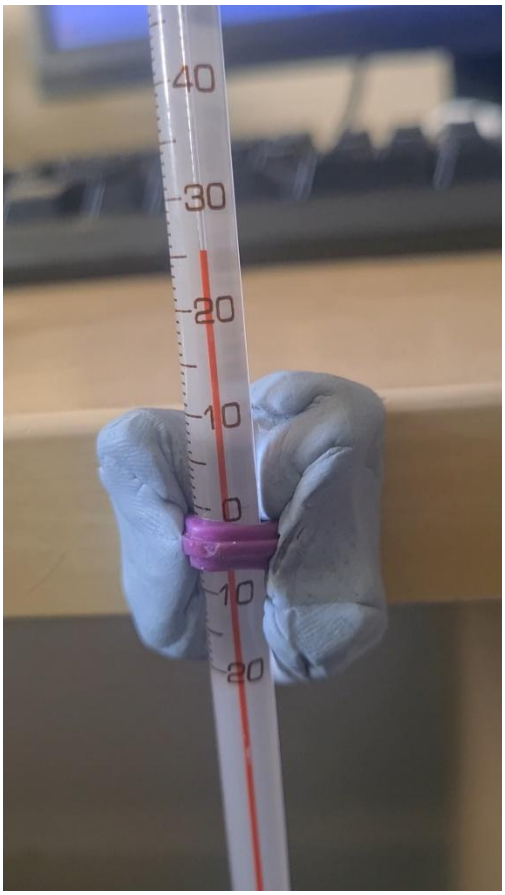
The aim of the Experimental Design is to find out the accuracy and testing the distance to ratio in which the sensor will reach, and the accuracy on e.g. different parameters such as long distance and short distances.

In addition to this, it will also measure the ability for the sensor to identify readings based on object dynamically changing positions (x y z) coordinates, and whether it reads data better if its static objects instead.

Initial Data –

The independent variables are the different lengths (distances) that the sensor will be measured and tested on. This will serve as a parameter for testing not only the quality, but the differences in quality of sensing different environmental variables from different distances (far away) and (close).

Experimental / Actual Result (Graphs and Tables) –



Discussion –

Conclusion –

Experiment B –

Investigation of patterns/correlations of Air Quality Parameters: Investigating Gas Concentrations (MQ Sensors) and Particulate Matter Levels (PM2.5), Infrared Thermal Camera (MLX90640), Temperature/Humidity (DHT11) with Time of Day in a (24 hour data reading) in relation to the Occupancy detected.

Aim –

The aim of the investigation is to identify the patterns/correlations of Air Quality Parameters is to identify any potential patterns, that repeat sequentially, over fixed periods of time (ranging from 24 hours to 1 week).

To carry out the investigation, we will be running the LSTM model, regression related model to analyse the dataset provided by ChatGPT, to identify future data based on Long-Term Short-Term Memory

Initial Data –

To test the effectiveness of the previous, we analysed it using a specific rate in which

The limitations of the DHT11 sensor, associated with humidity readings was impacted by the embedded number of decimal points allocated to the reading.

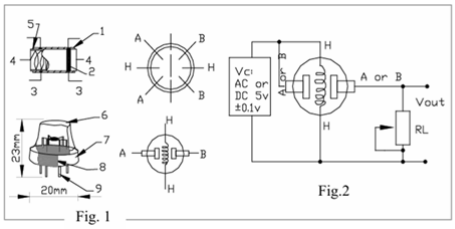


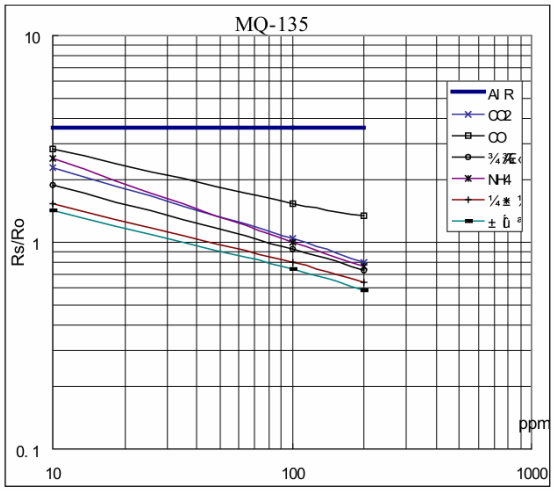
Fig. 1 Fig.2

In addition to this, we analysed the humidity every second, which gave us a knowledge gap – but can solved using interpolation in particularly when plotting the sensor readings in time series, (line graph).

Experimental / Actual Result (Graphs and Tables) -

Conclusion –

Discussion –



Experiment C

Effects of Bayesian Data Compression on CNN-based Occupancy Detection Accuracy

Aim

To investigate how data compression using Gaussian Naive Bayes-based uncertainty ranking affects the performance of a Conv1D-based CNN model for occupancy detection. This experiment evaluates whether reduced training data — selected based on Bayesian information gain — can maintain acceptable model accuracy while reducing computational overhead.

Initial Data

- Dataset: Merged environmental and motion sensor readings in .csv format
- Features: Temperature, Humidity, CO_2, TVOC, Light, Sound, Motion
- Original dataset size: ~N full rows
- Compression: Top 50% most uncertain instances selected using GNB's predict_proba output

Experimental / Actual Result (Graphs and Tables)

Discussion

Conclusion –

Hardware Implementation (Results)

Unfortunately, the deployment and full implementation of the CNN Conv1D neural network was not possible, due to the time constraints in place.

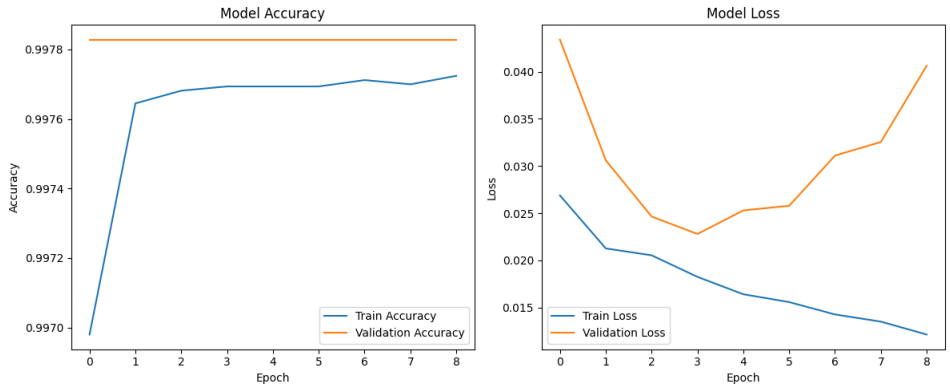
There was various hardware failures consisting of unreliable wiring, and lengthened learning time period required to find, and create the circuit diagram on the KiCad Schematic and reproduce this in a breadboarded prototype successfully – rather effectively.

The parts from China had to be resoldered, since the electrical pins were not pre-assembled and ready for prototyping/deployment.

All of the sensors, required adequate assembly and precise GPIO configurations in order to prevent any health and safety issues, e.g. burning the VCC pin when the power supply exceeds the sensors' allowed limits (per their datasheet)

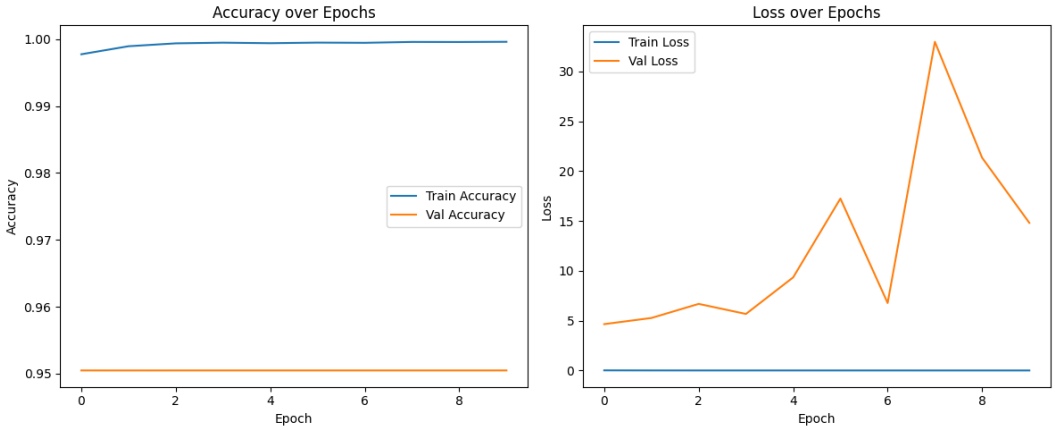
Software Implementation (Results)

The training of the Conv1D CNN model was successful, although using different datasets did present various interfacing issues. For instance, when using a large open-source dataset for occupancy and environmental monitoring purposes() – the data overfitted and as a result the model memorized rather than learned it regardless of the implementation of DropOut and EarlyStopping (as show in Fig.)



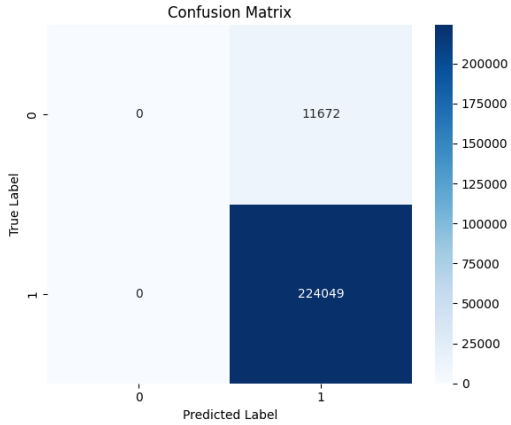
Upon initial attempts to run the program, it was evident that development without any prior respective libraries, e.g. **panda's** library led to various challenges in the long-term development of the programs, for instance the training script for the model development.

- Difficulties in interfacing with large libraries and utilizing precompiled binary wheels, as they allow for faster installation compared to building from source, particularly when attempting to install and use the **panda's** library.
- Issues related to updating the Python version, which hindered the installation and compatibility of the required libraries, as it was required for me to learn how to exclusively use the python3 version or update outdated version and libraries.



	precision	recall	f1-score	support
0.0	0.00	0.00	0.00	11672
1.0	0.95	1.00	0.97	224049
accuracy			0.95	235721
macro avg	0.48	0.50	0.49	235721

Accuracy:	0.9505
Precision:	0.9505
Recall:	1.0000
F1 Score:	0.9746
ROC-AUC:	0.5000
Log Loss:	0.7894



VIII. FUTURE WORK

Due to various time constraints, and time occupied working on various stages of the CNN model and waiting time for hardware delivery from China. It was not possible to create a simple GUI based web application, for instance the Adafruits Blinka Iot based Application. This could show the environmental data in real-time streamed to your iOS/Android smartphone device. Beyond that, by analyzing human behaviour like regular pauses or hyperactivity, the system has potentials of being extended to monitor and assist in user's welfare/mood patterns. In occupational health and smart office design, this might produce pretty interesting applications.

All things considered, future improvements are quite possible. Perhaps if there was available more time, better technology, and more testing would help this project to be really advanced.

Should it have succeeded, the technology would have been able to predict not only the occupancy of a room but also its population and location in real-time. Looking ahead, considering a better thermal sensor with more dependability and resolution, e.g. the FLIR Lepton camera might have been a worthwhile investment in the long run.

Developing a basic mobile or web application, very similar to Adafruits Blinka. Which could show the environmental data live would also be fantastic. Beyond that, the system might be extended to identify user welfare or mood patterns by examining behaviour, such as regular pauses or too lengthy of stillness. This might create quite fascinating applications in smart office design and occupational health.

All things considered, future enhancements have great possibility. More time, improved technology, and more testing would enable this project to be considerably advanced. Though the project's primary objectives were met—like accumulating environmental data and running local ML models—time,

technical, and testing constraints prevented me from doing some of the things I wanted to. Creating and testing an implementation of an original PCB manufacturered printout converted from the schematic diagram created on the KiCad was one of the objectives.

Manufacturing this would have greatly contributed to create a prototype more dependable, small, and appropriate for usage in a practical situation.

I also meant to build a wireless sensor network using ESP8266 modules or Zigbee connectivity but lacked time to sufficiently test or mix them. This would have made the system more scalable so that sensors could be placed in various rooms free from all the connections.

Another idea breaking from assumptions was occupancy counts using infrared imaging. Though I had set the MLX90640 thermal camera to identify individuals dependent on 2D thermal data, sadly it did not work as expected. Should it have been successful, the system might have forecast not only the occupancy of a room but also its population and location. Looking forward, the FLIR Lepton, a superior thermal sensor with higher dependability and resolution, could be well worth it.

X. CONCLUSION

XI. REFERENCES

[1] Dahane, A., Benameur, R. and Kechar, B., 2022. An IoT low-cost smart farming for enhancing irrigation efficiency of small-holder's farmers. Wireless Personal Communications, 127, pp. 3173–3210. https://doi.org/10.1007/s11277-022-09915-4.

[2] Xu, K., Zhang, H., Li, Y., Zhang, Y., Lai, R. and Liu, Y., 2023. An ultra-low power TinyML system for real-time visual processing at edge. IEEE Transactions on Circuits and Systems—II: Express Briefs, 70(7), pp. 2640–2641. https://doi.org/10.1109/TCSII.2023.3239044.

[3] Infineon Technologies, 2024. CY8C40x5, CY8C40x6 PSoC™ 4 MCU: PSoC™ 4000T datasheet based on Arm® Cortex®-M0+ CPU. Infineon. Available at: https://www.infineon.com [Accessed 28 November 2024].

[4] Jouini, O., Sethom, K., Namoun, A., Aljohani, N., Alanazi, M. H. and Alanazi, M. N., 2024. A survey of machine learning in edge computing: Techniques, frameworks, applications, issues, and research directions. Technologies, 12(81). https://doi.org/10.3390/technologies12060081.

[5] Shamim, M. Z. M., 2022. TinyML model for classifying hazardous volatile organic compounds using low-power embedded edge sensors: Perfecting Factory 5.0 using Edge AI. IEEE Sensors Letters, 6(9), p. 6003204. https://doi.org/10.1109/LSENS.2022.3201398.

[6] Zaidi, S. A. R., Hayajneh, A. M., Hafeez, M. and Ahmed, Q. Z., 2022. Unlocking edge intelligence through tiny machine learning (TinyML). IEEE Access, 10, pp. 100867–100868. https://doi.org/10.1109/ACCESS.2022.3207200.

[7] Sadooghi, I., Hernández Martin, J., Li, T., Brandstatter, K., Maheshwari, K., Pais Pitta de Lacerda Ruivo, T., Garzoglio, G., Timm, S., Zhao, Y. and Raicu, I., 2015. Understanding the performance and potential of cloud computing for scientific applications. IEEE Transactions on Cloud Computing, 5(2), pp. 358–374. https://doi.org/10.1109/TCC.2015.2404821.

[8] Berisha, B., Mëziu, E. and Shabani, I., 2022. Big data analytics in cloud computing: An overview. Journal of Cloud Computing: Advances, Systems and Applications, 11(24). https://doi.org/10.1186/s13677-022-00301-w.

[9] Mukhopadhyay, S., Dey, S., Ghose, A., Singh, P. and Dasgupta, P., 2023. Generating Tiny Deep Neural Networks for ECG Classification on Micro-Controllers. 2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), Atlanta, GA, USA, pp.392–397. doi: 10.1109/PerComWorkshops56833.2023.10150311.

[10] STL Partners , 2021. What is an Edge Server?. [YouTube Video] Available at: https://www.youtube.com/watch?v=_RKu9fBfvZg (Accessed: 3 January 2025).

[11] Torfs, T., Sterken, T., Brebels, S., Santana, J., van den Hoven, R., Spiering, V., Bertsch, N., Trapani, D., & Zonta, D. , 2013.. Low power wireless sensor network for building monitoring. IEEE Sensors Journal, 13(3), 909-915. https://doi.org/10.1109/JSEN.2012.2218680

[12] X. Hou, Y. Hu and F. Wang, "Overview of Task Offloading of Wireless Sensor Network in Edge Computing Environment," 2023 IEEE 6th International Conference on Electronic Information and Communication Technology (ICEICT), Qingdao, China, 2023, pp. 1018-1022, doi: 10.1109/ICEICT57916.2023.10245473.

[13] He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep residual learning for image recognition. arXiv preprint arXiv:1506.03099. https://arxiv.org/abs/1506.03099

[14] Tan, M., & Le, Q. , 2019.. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Available at: [efficientnet.pdf]

[15] Kam, K. H., Wong, C. S., Sor, K. V., Ooi, B. Y., & Hong, Z. W. , 2023.. Performance and Efficiency Comparison of VisionFive V1 Board and Raspberry Pi 4B. 2023 International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan). IEEE. DOI: 10.1109/ICCE-

Taiwan58799.2023.10226710. Available at: [Performance_and_Efficiency_Comparison_of_VisionFive_V1_Board_and_Raspberry_Pi_4B.pdf]

[16] Iandola, F., Han, S., Moskewicz, M., Ashraf, K., Dally, W. J., & Keutzer, K. , 2016.. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. Available at: [squeezenet.pdf]

[17] Signal Analysis Lab (n.d.) Industrial particulate monitoring. Available at: https://sal.no (Accessed: 18 February 2025).

[18] Dahane, A., Benameur, R. & Kechar, B. , 2022. "An IoT low-cost smart farming for enhancing irrigation efficiency of small-holder"s farmers', Wireless Personal Communications, 127, pp. 3173-3210. Available at: https://doi.org/10.1007/s11277-022-09915-4.

[19] Xu, K., Zhang, H., Li, Y., Zhang, Y., Lai, R. & Liu, Y. , 2023. "An ultra-low power TinyML system for real-time visual processing at edge", IEEE Transactions on Circuits and Systemsâ€"II: Express Briefs, 70(7), pp. 2640-2641. Available at: https://doi.org/10.1109/TCSII.2023.3239044.

[20] Jouini, O., Sethom, K., Namoun, A., Aljohani, N., Alanazi, M. H. & Alanazi, M. N. , 2024. "A survey of machine learning in edge computing: Techniques, frameworks, applications, issues, and research directions", Technologies, 12(81). Available at: https://doi.org/10.3390/technologies12060081.

[21] Shamim, M. Z. M. , 2022. "TinyML model for classifying hazardous volatile organic compounds using low-power embedded edge sensors: Perfecting Factory 5.0 using Edge AI", IEEE Sensors Letters, 6(9), p. 6003204. Available at: https://doi.org/10.1109/LSENS.2022.3201398.

[22] Zaidi, S. A. R., Hayajneh, A. M., Hafeez, M. & Ahmed, Q. Z. , 2022. "Unlocking edge intelligence through tiny machine learning (TinyML)", IEEE Access, 10, pp. 100867-100868. Available at: https://doi.org/10.1109/ACCESS.2022.3207200.

[23] Sadooghi, I., HernÃindez Martin, J., Li, T., Brandstatter, K., Maheshwari, K., Pais Pitta de Lacerda Ruivo, T., Garzoglio, G., Timm, S., Zhao, Y. & Raicu, I. , 2015. "Understanding the performance and potential of cloud computing for scientific applications", IEEE Transactions on Cloud Computing, 5(2), pp. 358-374. Available at: https://doi.org/10.1109/TCC.2015.2404821.

[24] Berisha, B., MÃ©ziu, E. & Shabani, I. , 2022. "Big data analytics in cloud computing: An overview", Journal of Cloud Computing:

[25] Gómez Larrakoetxea, N., Eskubi Astobiza, J., Pastor López, I., Sanz Urquijo, B., García Barruetabeña, J. and Zubillaga Rego, A., 2023. Efficient machine learning on edge computing through data compression techniques. IEEE Access, 11, pp. 31676–31677. https://doi.org/10.1109/ACCESS.2023.

[26] Freetronics (n.d.) OLED128 QuickStart Guide - Raspberry Pi. Available at: https://www.freetronics.com.au (Accessed: 5 December 2024).

[27] How2Electronics (n.d.) DIY Thermal Imaging Camera with MLX90640 & Raspberry Pi. Available at: https://how2electronics.com/diy-thermal-imaging-camera-with-mlx90640-raspberry-pi/ (Accessed: 5 December 2024).

[28] DataCamp , 2024.. Predictive Modelling for Agriculture. Available at: https://projects.datacamp.com/projects/1772 (Accessed: 10 December 2024)

[29] DataCamp , 2024.. Predictive Modelling for Environmental Modelling. In this project, a chatbot is used to assist in producing lightweight models. Available at: https://projects.datacamp.com/projects/1772 (Accessed: 10 December 2024).

[30] How2Electronics, , 2024.. Thermal Fever Detector with MLX90640, OpenCV & Raspberry Pi. Available at: https://how2electronics.com/thermal-fever-detector-with-mlx90640-opencv-raspberry-pi/ [Accessed 29 December 2024].

[31] Datacenters.com, 2023. AI and Machine Learning on Bare-Metal IaaS. [online] Available at: https://www.datacenters.com/news/ai-and-machine-learning-on-bare-metal-iaas? [Accessed 30 Dec. 2024].

[32] Dey, S., Mukherjee, A., Pal, A. and Balamuralidhar, P., 2019. Embedded Deep Inference in Practice: Case for Model Partitioning. In: 1st Workshop on Machine Learning on Edge in Sensor Systems (SenSys-ML 2019). New York, NY, USA, 10 November 2019. ACM. doi: 10.1145/3362743.3362964.

[33] Designing a Bare Minimum Face Recognition Architecture for Bare Metal Edge Devices: An Experience Report. [Further details required to complete this reference].

[34] van Wynsberghe, A., 2021. Sustainable AI: AI for sustainability and the sustainability of AI. AI Ethics, 1, pp.213–218. doi: 10.1007/s43681-021-00043-6.

[35] Raspberry Pi Foundation. , 2023. Raspberry Pi 5 product brief. Available at: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf (Accessed: 10/02/2026).

[36] Raspberry Pi Ltd. , 2023. Raspberry Pi Pico 2 datasheet. Available at: https://datasheets.raspberrypi.com/pico/pico-2-datasheet.pdf (Accessed: 10 February 2025).

[37] STMicroelectronics , 2025. CD00237391: Datasheet. Available at: https://www.st.com/resource/en/datasheet/cd00237391.pdf (Accessed: 10 February 2025).

[38] Raspberry Pi Ltd. , 2025. Raspberry Pi 5 Product Brief. Available at: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf (Accessed: 10 February 2025)

[39] Department for Environment, Food & Rural Affairs (DEFRA), 2024. UK-AIR: Air Information Resource - Data Catalogue. Available at: https://uk-air.defra.gov.uk/data/data-catalogue [Accessed 5 February 2025].

[40] Kalaiarasan, G., Kumar, P., Tomson, M., Zavala-Reyes, J. C., Porter, A. E., Young, G., Sephton, M. A., Abubakar-Waziri, H., Pain, C. C., Adcock, I. M., and Chung, K. F. , 2024.. Particle Number Size Distribution in Three Different Microenvironments of London. Atmosphere, 15(1), 45. https://doi.org/10.3390/atmos15010045

[41] Zayed, R. , 2022.. Big Data AI System for Air Quality Prediction. Conference Paper. Available at: https://www.researchgate.net/publication/377530969 [Accessed 11 Feb. 2025].

[42] World Health Organization, 2021. WHO global air quality guidelines: particulate matter (PM2. 5 and PM10), ozone, nitrogen dioxide, sulfur dioxide and carbon monoxide. World Health Organization.

[43] Rodríguez, E. H., Schalm, O. and Martínez, A., 2020. Development of a low-cost measuring system for the monitoring of environmental parameters that affect air quality for human health. ITEGAM-JETIA, 6(22), pp.22-27.

[44] Patel, Z. B., Bachwana, Y., Sharma, N., Guttikunda, S., & Batra, N. , 2024.. VayuBuddy: an LLM-Powered Chatbot to Democratize Air Quality Insights. arXiv preprint arXiv:2411.12760. Available at: https://arxiv.org/abs/2411.12760 [Accessed 11 Feb. 2025].

[45] Department for Environment, Food & Rural Affairs (Defra). (n.d.). PM2.5 Targets (PERT and AMCT). UK Air Information Resource. Available at: https://uk-air.defra.gov.uk/pm25targets/progress [Accessed 11 Feb. 2025].

[46] CapTech University, n.d. Ethical considerations of artificial intelligence. [online] CapTech University Blog. Available at: https://www.captechu.edu/blog/ethical-considerations-of-artificial-intelligence [Accessed 15 Feb 2025].

[47] Advances, Systems and Applications, 11(24). Available at: https://doi.org/10.1186/s13677-022-00301-w.

[48] Mukhopadhyay, S., Dey, S., Ghose, A., Singh, P. & Dasgupta, P. , 2023. "Generating Tiny Deep Neural Networks for ECG Classification on Micro-Controllers", 2023 IEEE International Conference on Pervasive Computing and Communications Workshops, Atlanta, GA, USA, pp. 392-397. Available at: https://doi.org/10.1109/PerComWorkshops56833.2023.10150311.

[49] Torfs, T., Sterken, T., Brebels, S., Santana, J., van den Hoven, R., Spiering, V., Bertsch, N., Trapani, D. & Zonta, D. , 2013. "Low power wireless sensor network for building

monitoring", IEEE Sensors Journal, 13(3), pp. 909-915. Available at: https://doi.org/10.1109/JSEN.2012.2218680.

[50] Hou, X., Hu, Y. & Wang, F. , 2023. "Overview of Task Offloading of Wireless Sensor Network in Edge Computing Environment", 2023 IEEE 6th International Conference on Electronic Information and Communication Technology (ICEICT), Qingdao, China, pp. 1018-1022. Available at: https://doi.org/10.1109/ICEICT57916.2023.10245473.

[51] He, K., Zhang, X., Ren, S. & Sun, J. , 2015. "Deep residual learning for image recognition", arXiv preprint, arXiv:1506.03099. Available at: https://arxiv.org/abs/1506.03099.

[52] Tan, M. & Le, Q. , 2019. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", Preprint (online). [Link to PDF or repository if available].

[53] Kam, K. H., Wong, C. S., Sor, K. V., Ooi, B. Y. & Hong, Z. W. , 2023. "Performance and Efficiency Comparison of VisionFive V1 Board and Raspberry Pi 4B", 2023 International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan). Available at: https://doi.org/10.1109/ICCE-Taiwan58799.2023.10226710.

[54] Iandola, F., Han, S., Moskewicz, M., Ashraf, K., Dally, W. J. & Keutzer, K. , 2016. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size", Preprint (online). [Link to PDF or repository if available].

[55] GÃ³mez Larrakoetxea, N., Eskubi Astobiza, J., Pastor LÃ³pez, I., Sanz Urquijo, B., GarcÃ¬a BarruetabeÃ±a, J. & Zubillaga Rego, A. , 2023. "Efficient machine learning on edge computing through data compression techniques", IEEE Access, 11, pp. 31676-31677. Available at: https://doi.org/10.1109/ACCESS.2023.

[56] Dey, S., Mukherjee, A., Pal, A. & Balamuralidhar, P. , 2019. "Embedded Deep Inference in Practice: Case for Model Partitioning", 1st Workshop on Machine Learning on Edge in

Sensor Systems (SenSys-ML 2019), New York, NY, USA, 10 November 2019. Available at: https://doi.org/10.1145/3362743.3362964.

[57] van Wynsberghe, A. , 2021. "Sustainable AI: AI for sustainability and the sustainability of AI", AI Ethics, 1, pp. 213-218. Available at: https://doi.org/10.1007/s43681-021-00043-6.

[58] Zayed, R. , 2022. "Big Data AI System for Air Quality Prediction", Conference paper (available on ResearchGate). Available at: https://www.researchgate.net/publication/377530969.

[59] R. Santos, "Raspberry Pi with DHT11 Temperature and Humidity Sensor using Python," Random Nerd Tutorials, Sep. 2020. [Online]. Available: https://randomnerdtutorials.com/raspberry-pi-dht11-dht22-python/

[60] "How to Use Gas Sensor with Raspberry Pi," Newbiely, [Online]. Available: https://www.newbiely.com/tutorials/raspberry-pi/raspberry-pi-gas-sensor. [Accessed: Apr. 10, 2025].

[61] A. S. Dey, M. J. Barth, and R. Dey, "HeteroSense: Occupancy Sensing Using Heterogeneous Data," in Proc. 6th ACM Int. Conf. on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '19), New York, NY, USA, Nov. 2019, pp. 265–266. [Online]. Available: https://dl.acm.org/doi/10.1145/3360322.3360824

[62] Melexis Technologies, "MLX90640 32×24 Infrared Array Datasheet," [Online]. Available: https://www.melexis.com/en/documents/documentation/datasheets/datasheet-mlx90640

[63] Adafruit Industries, "Adafruit MLX90640 Thermal Camera Python Library," [Online]. Available: https://github.com/adafruit/Adafruit_CircuitPython_MLX90640

[64] Adafruit Industries, "Adafruit DHT Sensor Library," [Online]. Available: https://github.com/adafruit/Adafruit_Python_DHT

[65] Adafruit Learning System, "Analog Sensors with Raspberry Pi using MCP3008," [Online]. Available: https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters/mcp3008

[66] Winsen Electronics, "MQ-135 Gas Sensor Datasheet," [Online]. Available: https://www.winsen-sensor.com/d/files/air-quality-sensor/mq135-gas-sensor-manual-v1.3.pdf

[67] Thonny IDE, "Python IDE for Beginners," [Online]. Available: https://thonny.org

[68] Python Software Foundation, "Python 3.11 Documentation," [Online]. Available: https://docs.python.org/3.11/

[69] S. Sarkar, M. Jacoby, G. Henze, and S. Y. Tan, *A High-Fidelity Residential Building Occupancy Detection Dataset*, figshare, 2021. [Online]. Available: https://doi.org/10.6084/m9.figshare.c.5364449.v1

[70] R. K. Srivastava, A. K. Jain, and A. Kumar, "Seasonal variation of indoor and outdoor particle number concentration in different socioeconomic household conditions in Delhi," Environmental Pollution, vol. 300, p. 118956, 2022. Available: https://doi.org/10.1016/j.envpol.2022.118956

[71] A. N. Sayed, F. Bensaali, Y. Himeur, and M. Houchati, "Edge-based real-time occupancy detection system through a non-intrusive sensing system," Energies, vol. 16, no. 5, p. 2388, Mar. 2023, doi: https://doi.org/10.3390/en16052388.

XII. APPENDIX

Legal Compliances of Development

- **BS ISO/IEC 24029-2:2023**

  Artificial intelligence (AI). Assessment of the robustness of neural networks. Methodology for the use of formal method
- **PD CEN/CLC/TR 18115:2024**

  Data governance and quality for AI within the European context
- **PD CLC/TS 50491-7:2024**

  General requirements for Home and Building Electronic Systems (HBES) and Building Automation and Control Systems (BACS). IT security and data protection.
- **BS ISO/IEC 25040:2024**

  Systems and software engineering. Systems and software Quality Requirements and Evaluation (SQuaRE). Quality evaluation framework
- **BS EN 61010-1 and BS EN 50470-1:**

  Ensure electrical safety and proper integration of environmental sensors.
- **BS ISO/IEC 27001**

  For securing data and information management.
- **BS EN ISO 8000**
- **BS EN ISO 14001**

  For ensuring that your project aligns with environmental management principles.
- **BS EN 61326-1**

  For ensuring that your real-time systems are compliant with EMC requirements.

- Tillaart, R. (n.d.) **DHT20 Sensor Library**. Available at: https://github.com/RobTillaart/DHT20/blob/master/LICENSE (Accessed: 20 December 2024).
- Adafruit (n.d.) **Adafruit MLX90640 Library**. Available at: https://github.com/adafruit/Adafruit_MLX90640/blob/master/LICENSE (Accessed: 20 December 2024).
- **STMicroelectronics** (2025) *CD00237391: Datasheet*. Available at: https://www.st.com/resource/en/datasheet/cd00237391.pdf (Accessed: 10 February 2025).
- **Raspberry Pi Ltd**. (2025) *Raspberry Pi 5 Product Brief*. Available at: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf (Accessed: 10 February 2025)

Ensuring the quality of collected sensor data for machine learning applications.

- **BS EN 300 328**

  For ensuring compliance with wireless communication standards, if using wireless sensors.

  - IPC, 2023. *IPC-2221: Generic standard on printed board design*. 12 January. Bannockburn, IL: IPC. ISBN 978-1-63816-150-9.
  - IPC, 2020. *IPC-2222: Sectional design standard for rigid organic printed boards*. Bannockburn, IL: IPC.
  - List of Tables on Literature Review

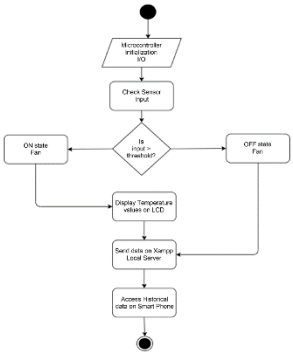
- **STMicroelectronics**, 2021. X-CUBE-TOUCHGFX: TouchGFX optimized graphic framework software expansion for STM32Cube. Product Datasheet. Available at: https://www.st.com [Accessed 11 Feb. 2025].
- **STMicroelectronics** (2023) *STM32H742xI/G & STM32H743xI/G Datasheet* (DS12110 Rev 10, March 2023). Available at: **https://www.st.com/resource/en/datasheet/stm32h743vi.pdf** (Accessed: 2 March 2025).
- **Texas Instruments**, 2012. TCA9548A Low-Voltage 8-Channel I2C Switch with Reset datasheet (Rev. G). Texas Instruments Technical Datasheet. Available at: https://www.ti.com [Accessed 11 Feb. 2025].
- **Melexis,** 2024. MLX90640 32x24 IR array. Product Datasheet. Available at: https://github.com/melexis/mlx90640- [Accessed 11 Feb. 2025].
- **DFRobot,** 2024. 7” 800x480 TFT DSI Capacitive Touchscreen. Product Datasheet. Available at: https://www.mouser.com/new/dfrobot/dfrobot-7-tft-dsi-capacitive-touchscreen/ [Accessed 11 Feb. 2025].

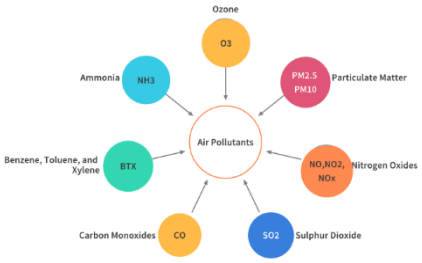
- Apvrille, L., 2024. UMLEmb: **UML for Embedded Systems II**. Modelling in SysML. Technical Report.
- Anderson, R., 2008. Security engineering: a guide to building dependable distributed systems. Wiley. ISBN: 9780470068526.

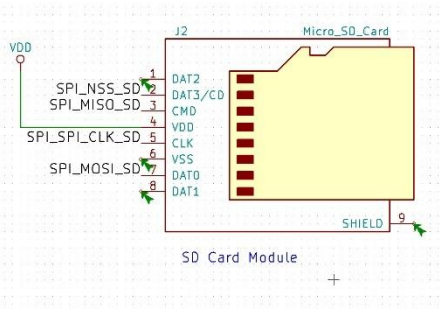
Abbreviations

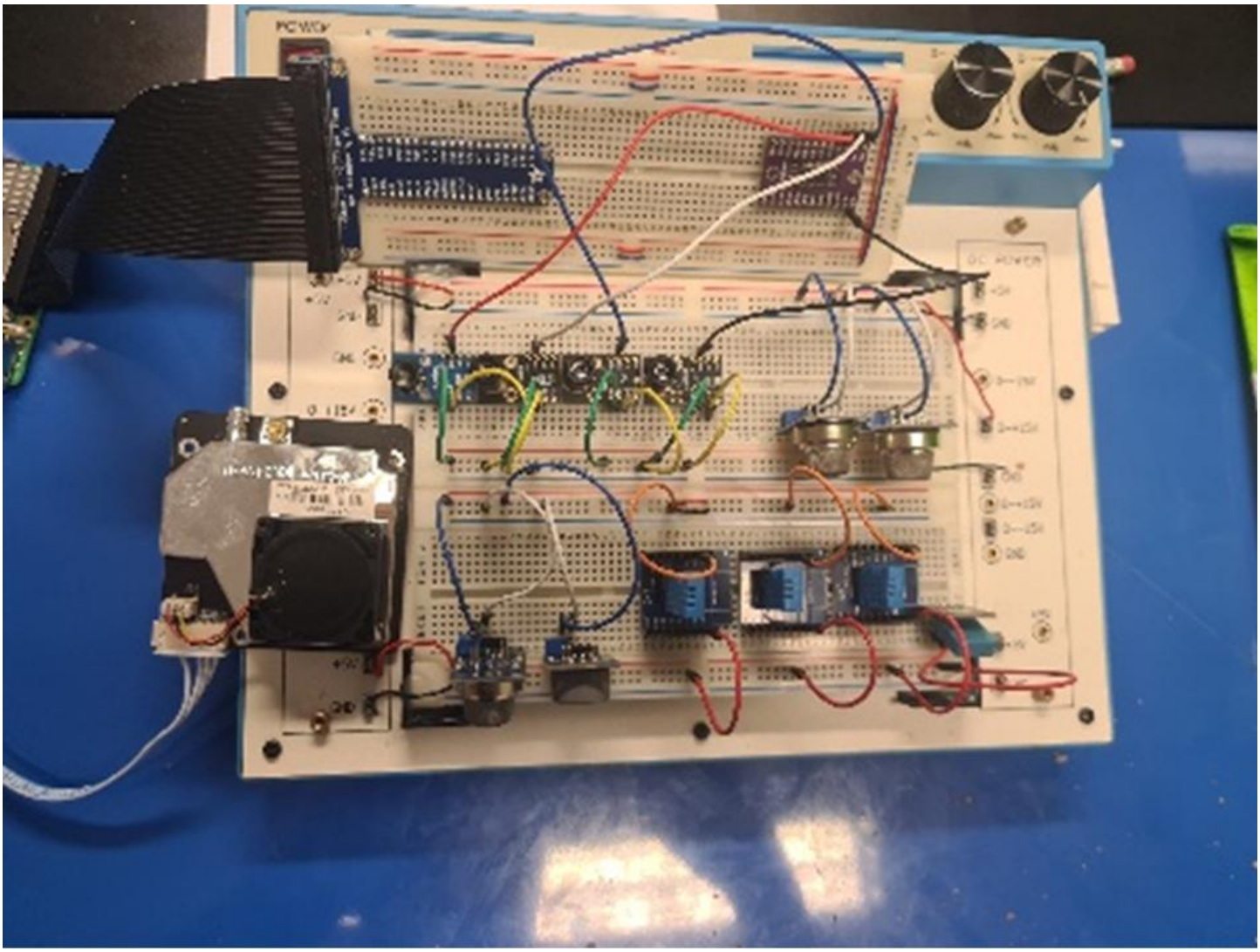
- **MQTT – Message Queuing Telemetry Transport**
- **HTTP – Hyper Text Transport Protocol**

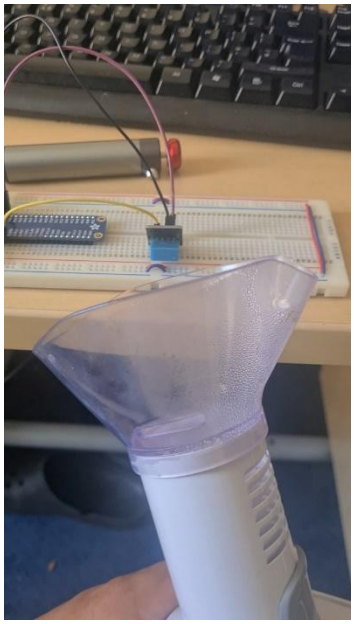
List of Figures

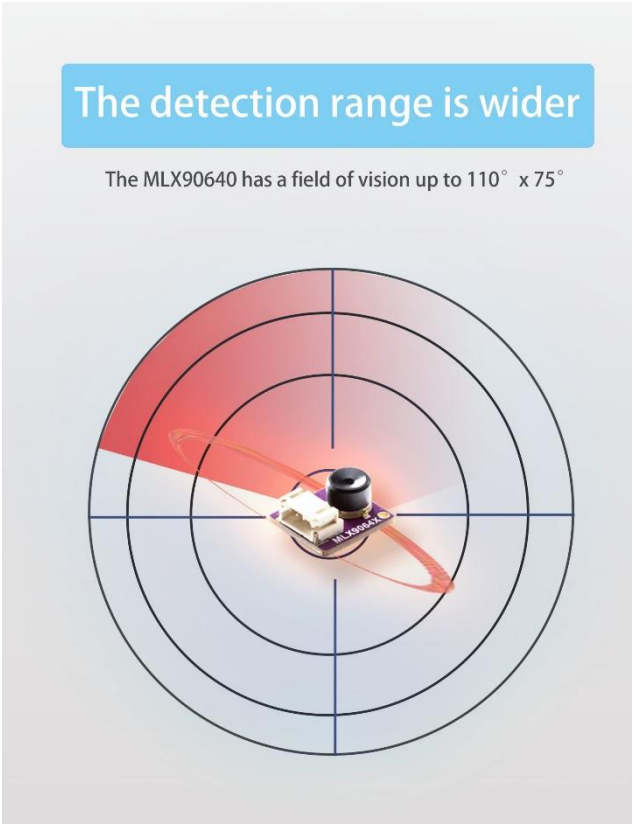


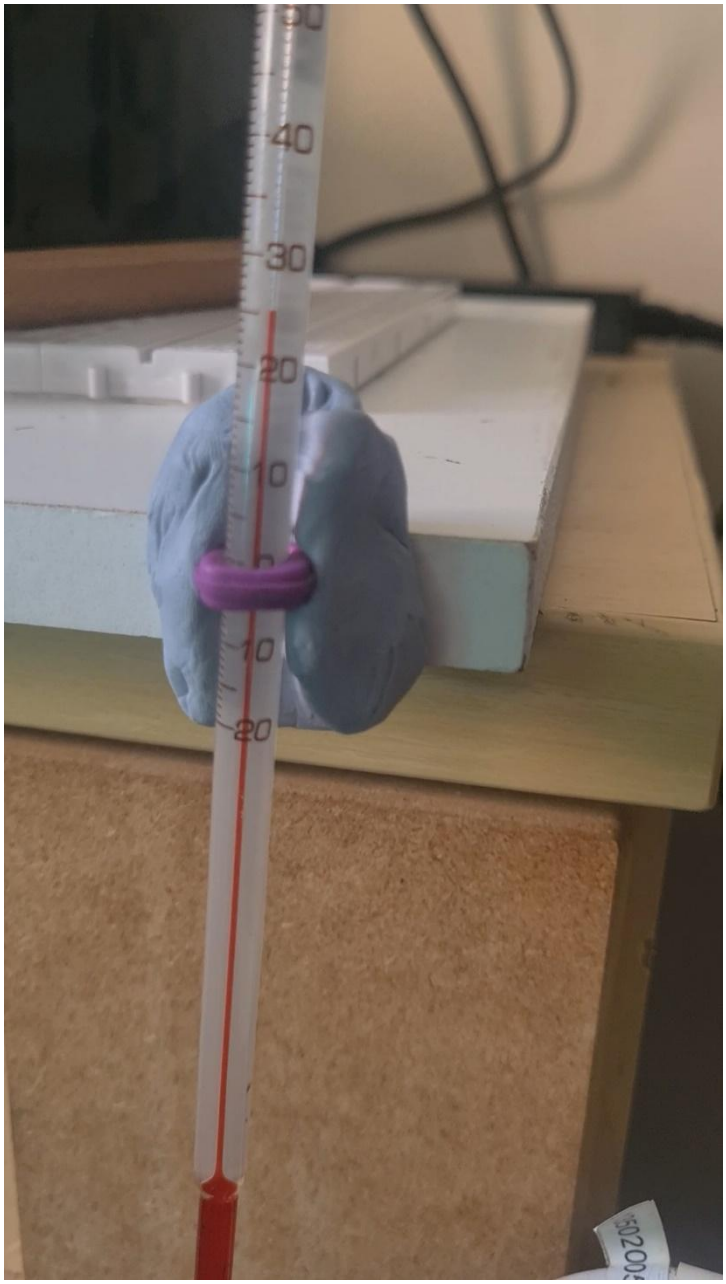


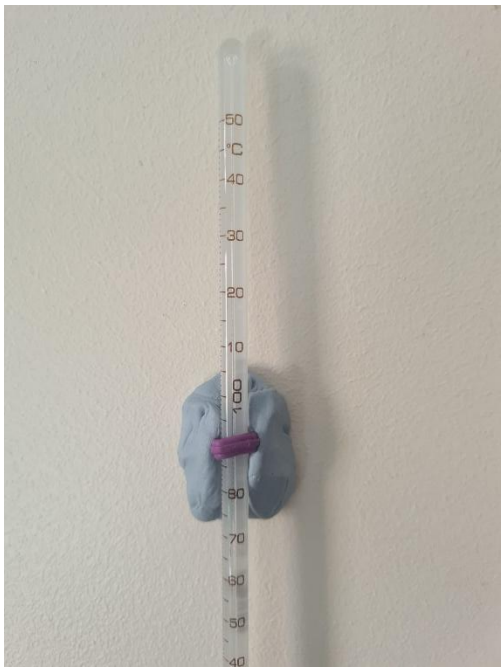


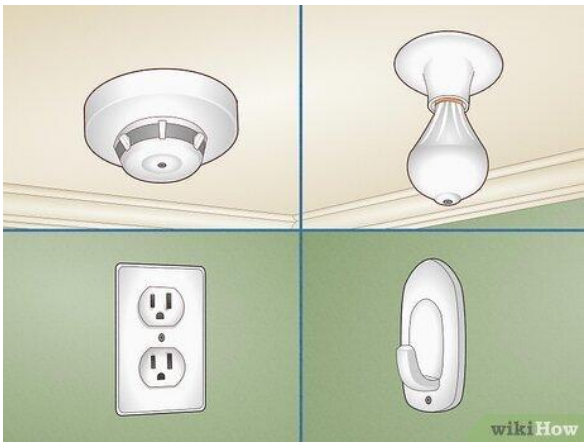


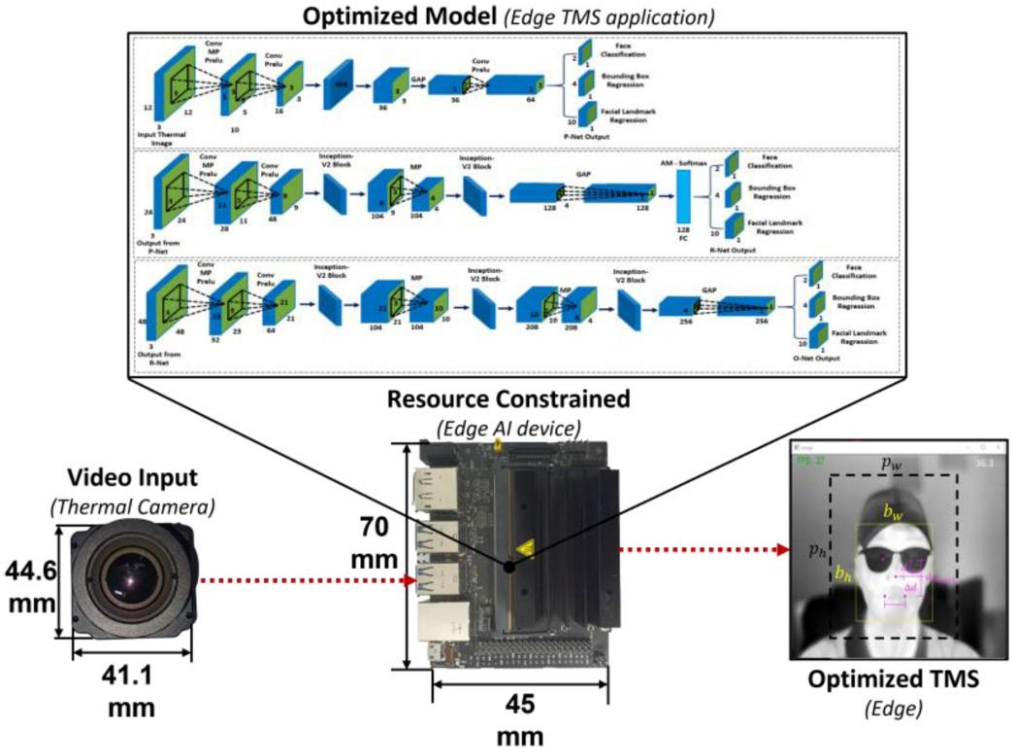


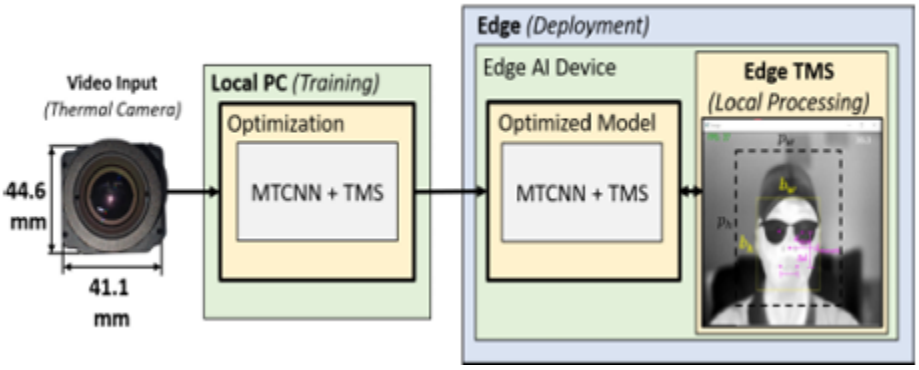

The detection range is wider
The MLX90640 has a field of vision up to 110° x 75°


40
30
20
10
10
20


50
°C
40
30
20
10
0
10
20
30
40
50
60
70
80
90
100


wikiHow


Optimized Model (Edge TMS application)
Resource Constrained
(Edge AI device)
Video Input
(Thermal Camera)
44.6
mm
41.1
mm
70
mm
45
mm
Optimized TMS
(Edge)


Video Input
(Thermal Camera)
44.6
mm
41.1
mm
Local PC (Training)
Optimization
MTCNN + TMS
Edge (Deployment)
Edge AI Device
Optimized Model
MTCNN + TMS
Edge TMS
(Local Processing)


1
Project setup
1. Understand the business goals
Speak with your stakeholders and deeply understand the business goal behind the model being proposed. A deep understanding of your business goals will help you scope the necessary technical solution, data sources to be collected, how to evaluate model performance, and more.
2. Choose the solution to your problem
Once you have a deep understanding of your problem—focus on which category of models drives the highest impact. See this Machine Learning Cheat Sheet for more information.
2
Data preparation
1. Data collection
Collect all the data you need for your models, whether from your own organization, public or paid sources.
2. Data cleaning
Turn the messy raw data into clean, tidy data ready for analysis. Check out this data cleaning checklist for a primer on data cleaning.
3. Split the data
Randomly divide the records in the dataset into a training set and a testing set. For a more reliable assessment of model performance, generate multiple training and testing sets using cross-validation.
4. Feature engineering
Manipulate the datasets to create variables (features) that improve your model's prediction accuracy. Create the same features in both the training set and the testing set.
3
Modeling
1. Hyperparameter tuning
For each model, use hyperparameter tuning techniques to improve model performance.
2. Train your models
Fit each model to the training set.
3. Make predictions
Make predictions on the testing set.
4. Assess model performance
For each model, calculate performance metrics on the testing set such as accuracy, recall and precision.
1
Project setup
1. Understand the business goals
Speak with your stakeholders and deeply understand the business goal behind the model being proposed. A deep understanding of your business goals will help you scope the necessary technical solution, data sources to be collected, how to evaluate model performance, and more.
2. Choose the solution to your problem
Once you have a deep understanding of your problem—focus on which category of models drives the highest impact. See this Machine Learning Cheat Sheet for more information.

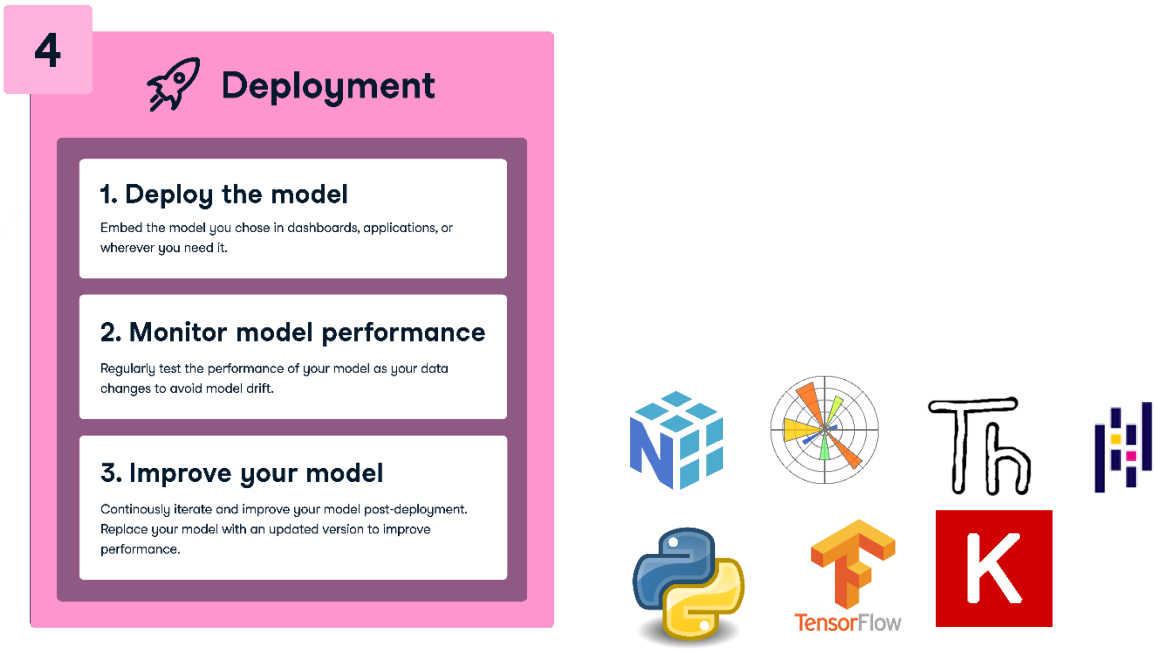


Fig. A1

Fig. Z

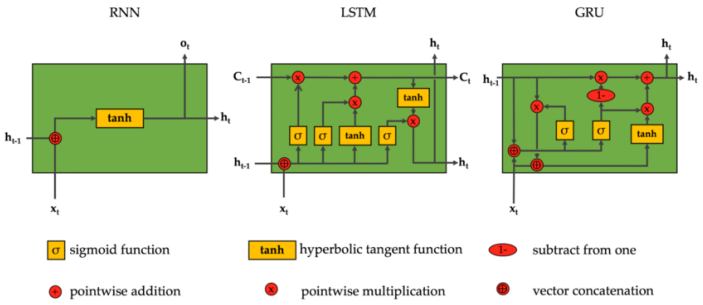


Fig. Y

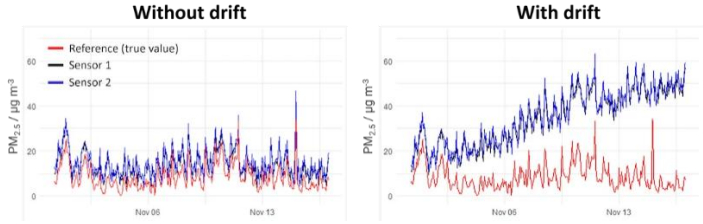


Fig. Y

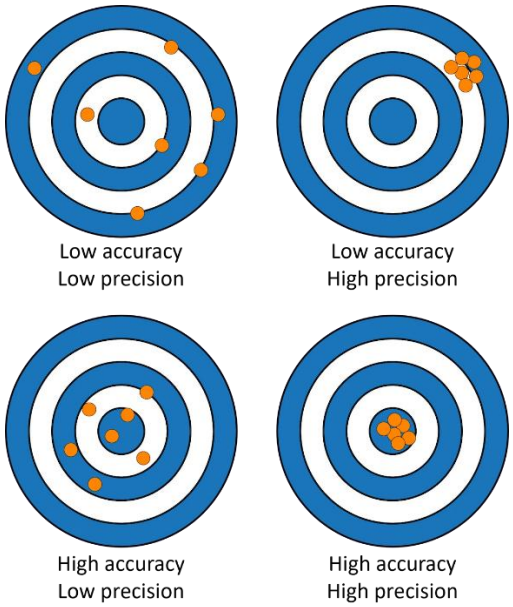


Fig. X

	Timestamp	Temperature (°C)	Humidity (%)	PM2.5
2	26/02/2025 00:00	22.75	50.41	36.03023740689505
3	26/02/2025 00:00	22.76	49.98	30.34876205884385
4	26/02/2025 00:01	22.76	49.55	29.501571054401207
5	26/02/2025 00:01	22.56	50.03	41.093769753332396
6	26/02/2025 00:02	22.35	50.51	30.278837270206132
7	26/02/2025 00:02	22.56	49.91	26.54219611709299
8	26/02/2025 00:03	22.76	49.31	26.543954367225577
9	26/02/2025 00:03	22.52	49.75	32.54400238744395
10	26/02/2025 00:04	22.29	50.19	28.106569551237428
11	26/02/2025 00:04	22.47	50.08	27.244523767032234
12	26/02/2025 00:05	22.65	49.97	24.275646447461853
13	26/02/2025 00:05	22.63	50.4	32.27341916391527
14	26/02/2025 00:06	22.61	50.84	31.144969347407354
15	26/02/2025 00:06	22.46	50.76	30.633469631752078
16		22.31	50.67	24.053895685393748
17		22.42	50.38	31.68959743185896
		52	50.08	25.5615416004
			50.37	30.562409
			50.66	27.19

Fig. W

Fig. V

Fig. U

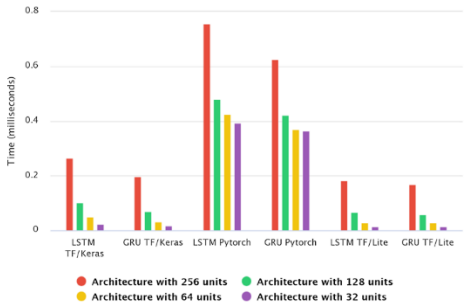


Fig. T

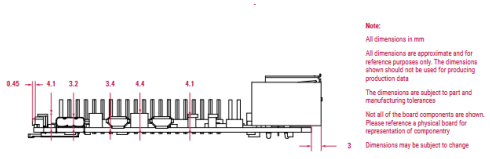


Fig. S

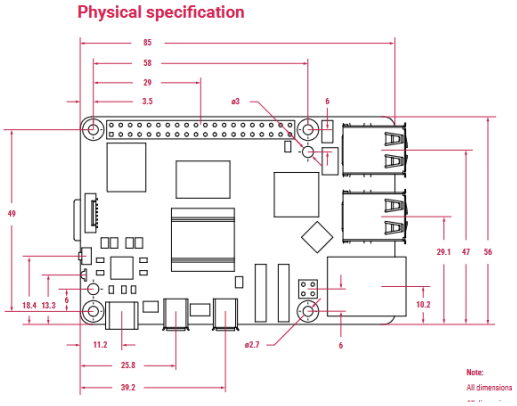


Fig. R

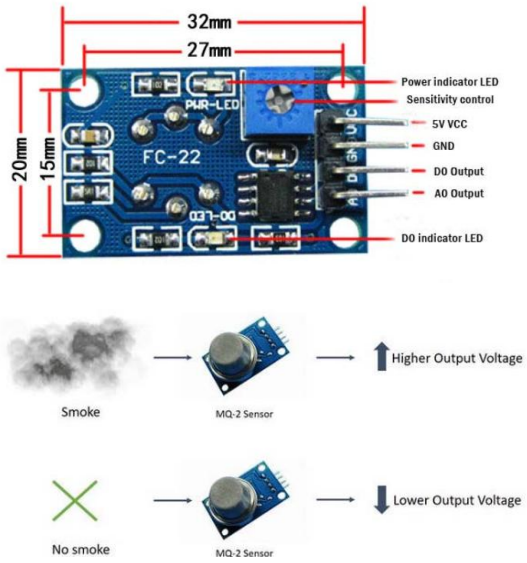


Fig. Q


°C Temperature °C/°F
% Humidity
> M T W T F S S (Non Working Days Inc.)
> Time (Working Hours incl.)
> Holiday / Term Periods
> Occupancy of Rooms
> Seminars / Large Event Meeting
> Area most visited
Contextual Data
+
> HIGH
> MEDIUM (NORMAL) +/-
> LOW
-
Categorical Encoding

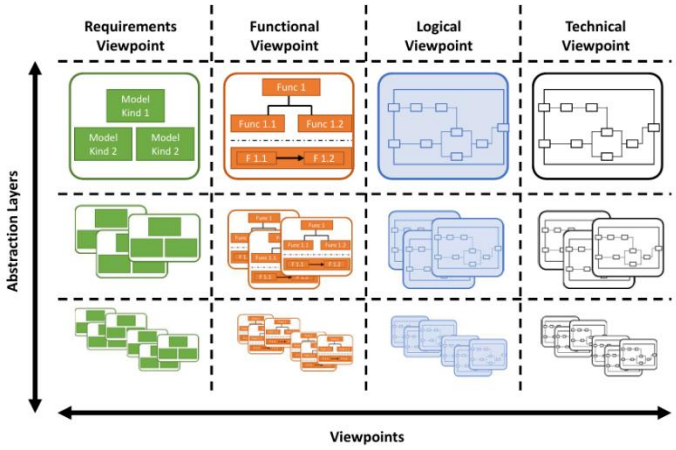

Requirements Viewpoint
Functional Viewpoint
Logical Viewpoint
Technical Viewpoint
Model Kind 1
Model Kind 2
Model Kind 2
Func 1
Func 1.1
Func 1.2
F 1.1
F 1.2
Abstraction Layers
Viewpoints

Fig. P

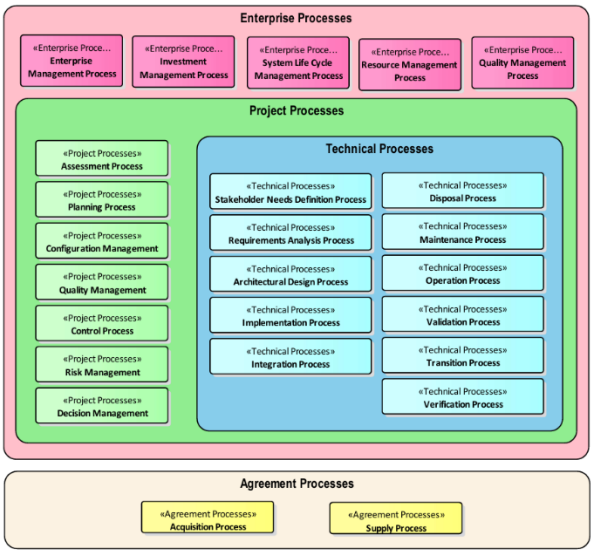

Enterprise Processes
«Enterprise Proce... Enterprise Management Process
«Enterprise Proce... Investment Management Process
«Enterprise Proce... System Life Cycle Management Process
«Enterprise Proce... Resource Management Process
«Enterprise Proce... Quality Management Process
Project Processes
«Project Processes» Assessment Process
«Project Processes» Planning Process
«Project Processes» Configuration Management
«Project Processes» Quality Management
«Project Processes» Control Process
«Project Processes» Risk Management
«Project Processes» Decision Management
Technical Processes
«Technical Processes» Stakeholder Needs Definition Process
«Technical Processes» Requirements Analysis Process
«Technical Processes» Architectural Design Process
«Technical Processes» Implementation Process
«Technical Processes» Integration Process
«Technical Processes» Disposal Process
«Technical Processes» Maintenance Process
«Technical Processes» Operation Process
«Technical Processes» Validation Process
«Technical Processes» Transition Process
«Technical Processes» Verification Process
Agreement Processes
«Agreement Processes» Acquisition Process
«Agreement Processes» Supply Process

Fig. O

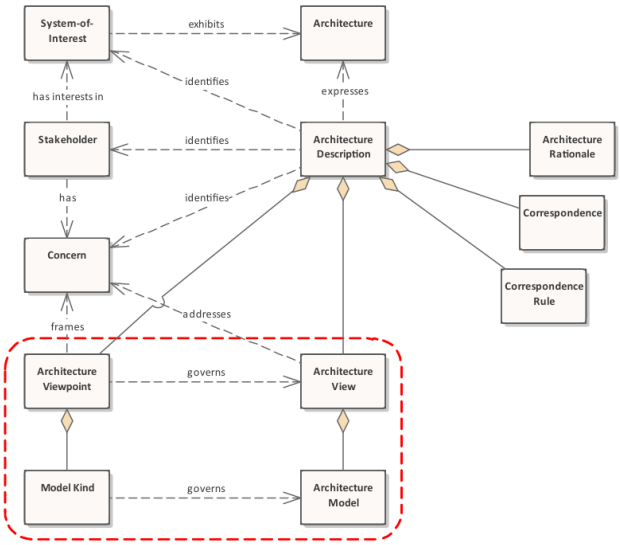

System-of-Interest
exhibits
Architecture
identifies
has interests in
expresses
Stakeholder
identifies
Architecture Description
Architecture Rationale
has
identifies
Correspondence
Concern
Correspondence Rule
frames
addresses
Architecture Viewpoint
governs
Architecture View
Model Kind
governs
Architecture Model

Fig. N

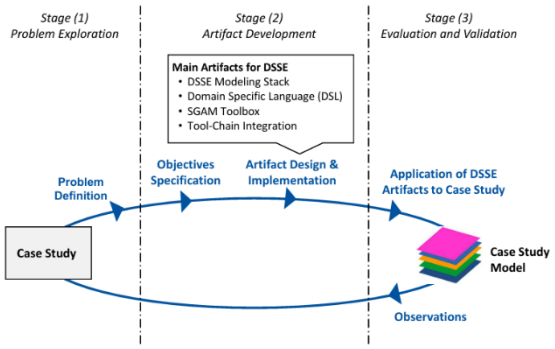


Figure 5. Application of ADSRM.

Fig. M

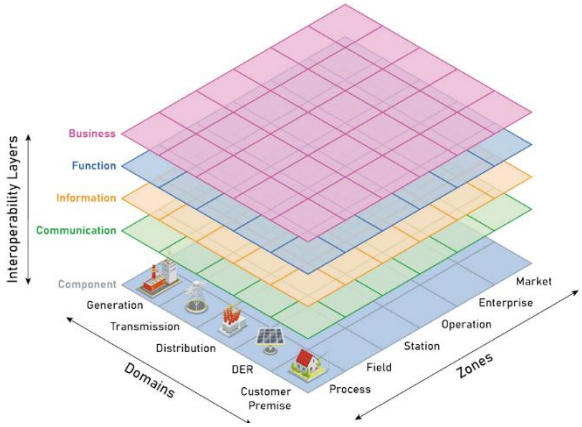


Fig. L

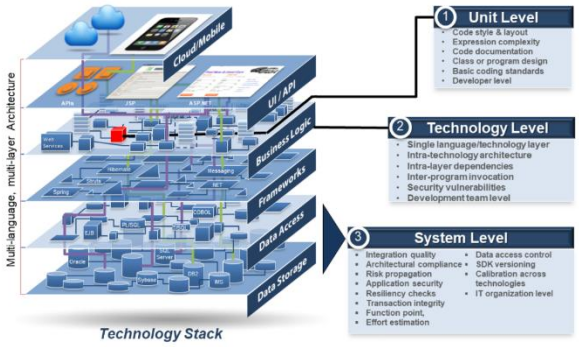


Fig. K

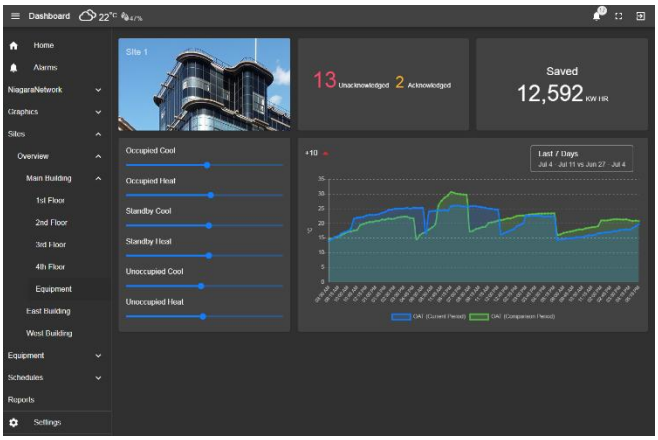


Fig. j

Fig. I

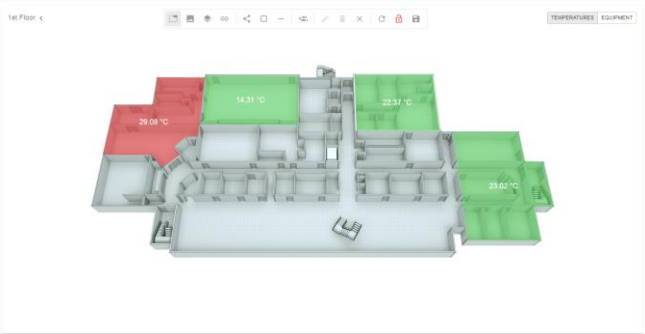


Fig. R

Fig. Q

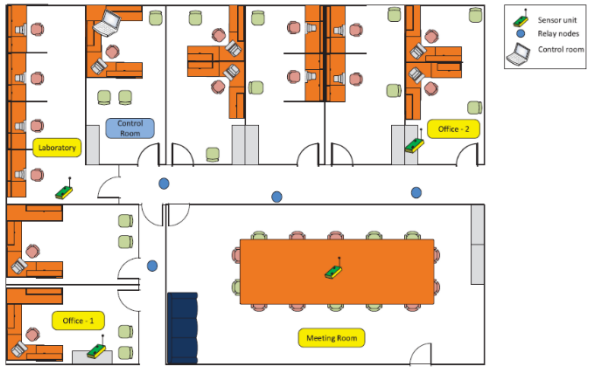


Fig. P

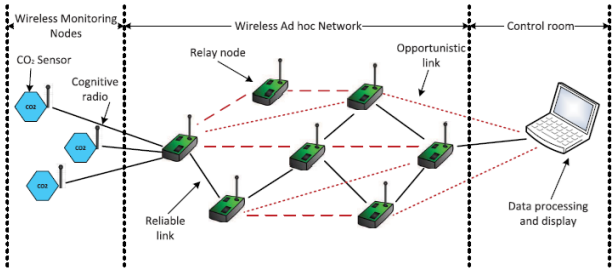


Fig. N

Fig. M

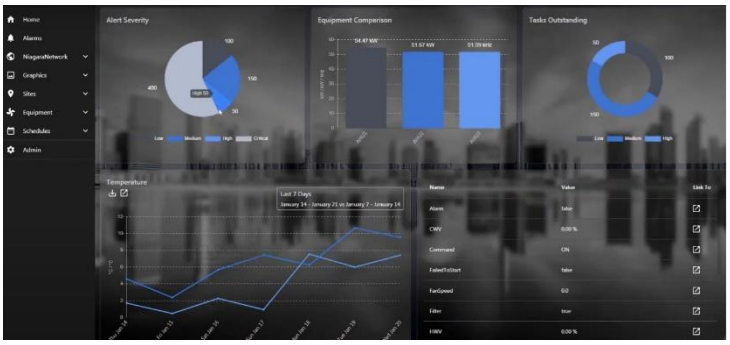


Fig. L

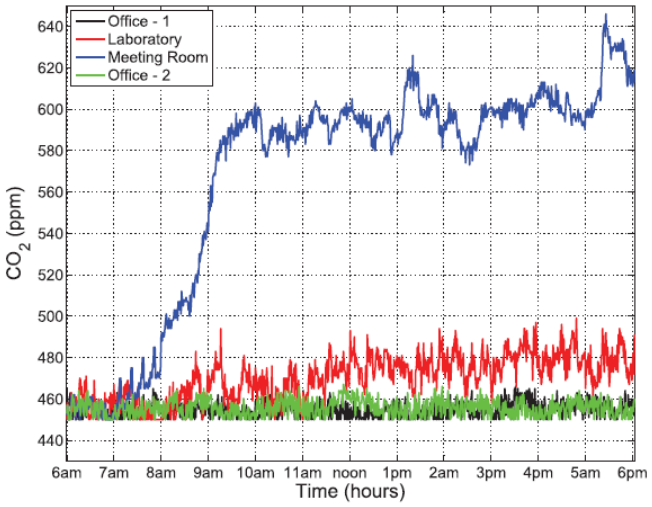


Fig. K

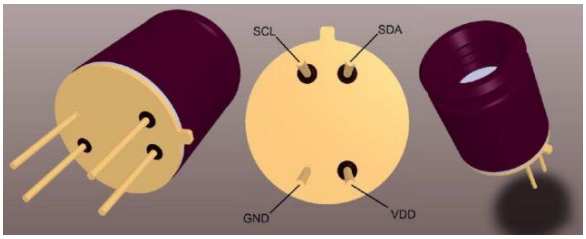


Fig. J

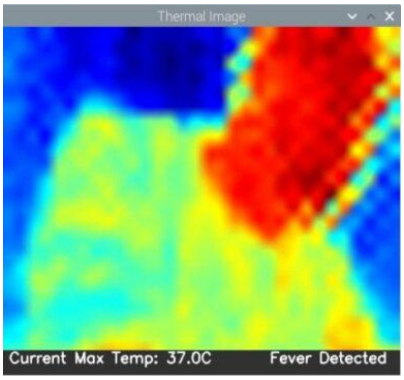


Fig. G

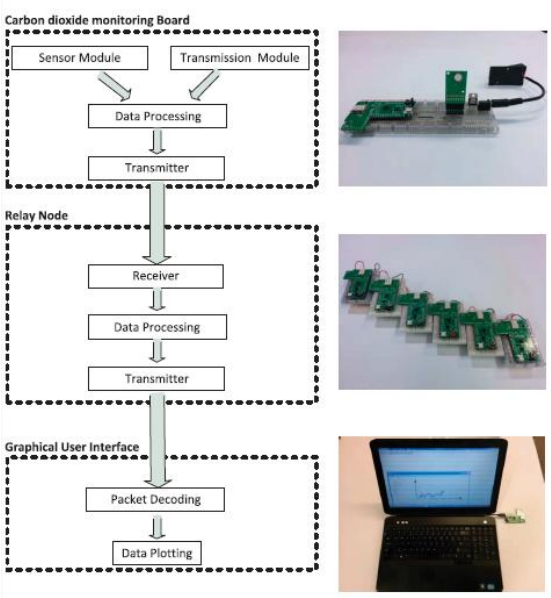


Fig. H

Fig. F

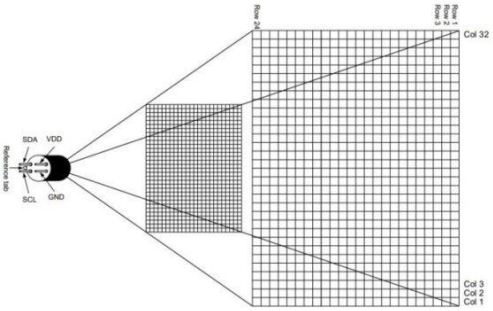


Fig. I

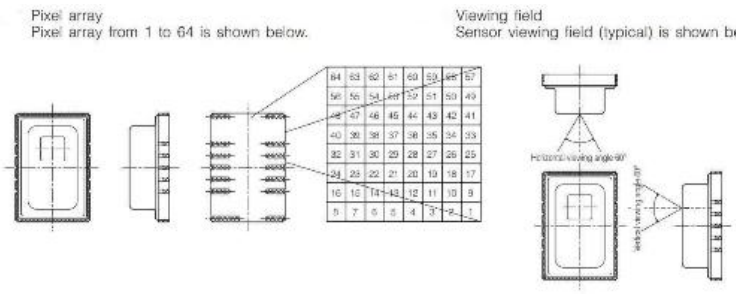


Fig. E

Reading ID	Humidity rate	Temperature in °C
0	0.89	7.388889
1	0.86	7.227778
2	0.89	9.377778
3	0.83	5.944444
...	...	...

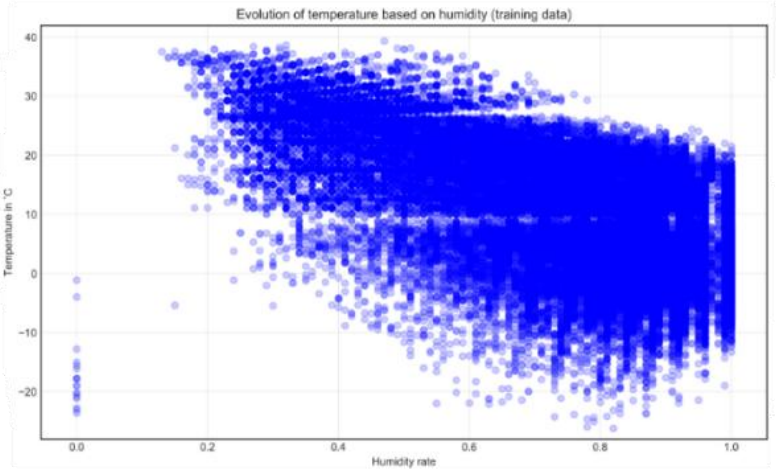


Fig. D

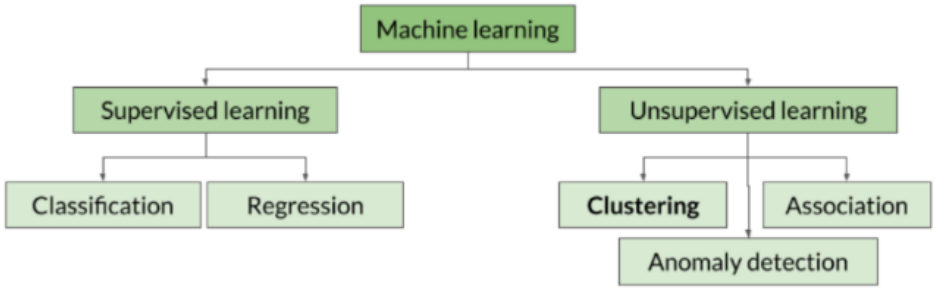


Fig. C

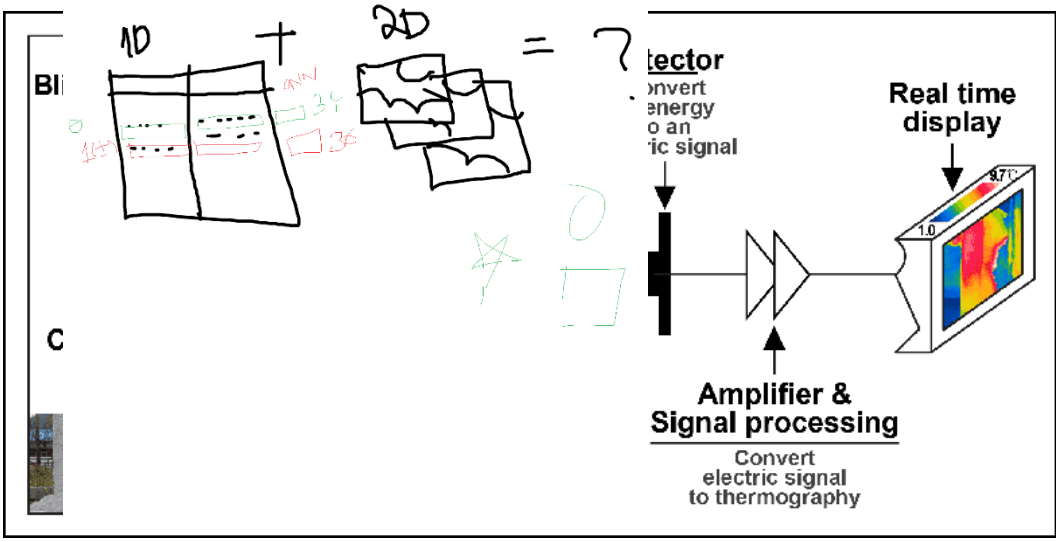


Fig. B

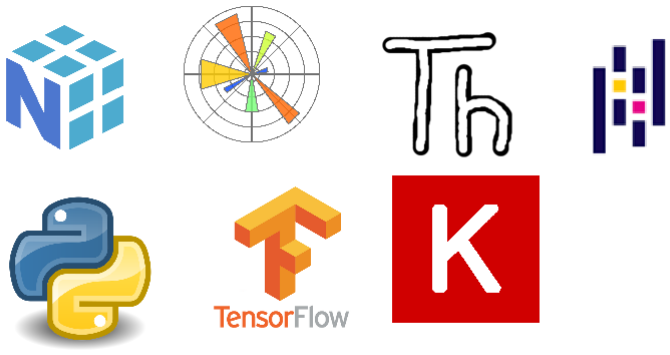


Fig. B

List of Tables

Relative humidity

parameter	condition	min	typical	max	unit
resolution	typical		0.024		%RH
Accuracy Error[1]	typical		±3		%RH
	max	See Figure 2			%RH
repeatable			±0.1		%RH
hysteresis			±1		%RH
nonlinear			<0.1		%RH
Response time[2]	t 63%		8		S
Scope of work	Extended[3]	0		100	%RH
Prolonged Drift[4]	normal		<0.5		%RH/yr

Table 1 Humidity characteristics table

Temperature

parameter	condition	min	typical	max	unit
resolution	typical		0.01		℃
Accuracy error[1]	typical		±0.5		℃
	max	See Figure 3			℃
repeatable			±0.1		℃
hysteresis			±0.1		℃
nonlinear					℃
Response time	t 63%	5		30	S
Scope of work	Extended[3]	-40		80	℃
Prolonged Drift			<0.04		℃/yr

Table 3 Temperature Characteristic table

Table. C

Specification	Raspberry Pi Model 4B (General edge architecture)	NVIDIA Jetson TX2 (Heterogeneous edge architecture)	Amazon EC2 t2.xlarge (General cloud architecture)
CPU	A quad-core cortex-A72 ARM v8 64-bit SoC 1.5GHz	A quad-core 2.0GHz 64-bit ARM v8 A57 processor; a dual-core 2.0GHz superscalar ARM v8 Denver processor	A quad-core 3.0GHz Intel scalable processor (vCPU)
GPU	No GPU	56-core 1.33 TFLOPS NVIDIA Pascal	No GPU
Cache	32 KB L1 data cache 48 KB L1 instruction cache 1MB L2 cache	64KB L1 data cache 128KB L1 instruction cache 2MB L2 cache	32KB L1 data cache 32KB L1 instruction cache 256KB L2 cache

Table. B

Table 2. *Cont.*

Paper	Application Domain	ML Model	Framework / Hardware	Architecture	Benefits	Drawbacks
[75]	Smart Agriculture	LSTM and Encoder Decoder	TensorFlow with Arduino (Mkr and Uno), Raspberry Pi 4 Model B and NVIDIA Jetson Nano	Edge Layer Computing	Higher Prediction and Low Time Consumption	High Complexity in Real-world Deployment
[76]	Smart Agriculture	Random Forest	Raspberry Pi	Edge Layer Computing	High Accuracy and Reduced Data Transmission Costs and Latency	Limited Scalability
[88]	Smart Environment	CNN	Caffe with Intel Core i7 7770 CPU and NVIDIA Geforce GTX 1080 graphic card	Distributed Edge and Cloud Computing	Enhanced Privacy and Reduced Network Traffic	High Computational Cost and Limited Scalability
[89]	Smart Agriculture	H20	Drone enabled with a Tetracam ADC lite camera	Distributed Edge and Cloud Computing	Reduced Security Risks and Low latency Compared to Centralized	High Computational Cost
[90]	Surveillance Systems	SVM	Wireless camera on a Raspberry Pi	Distributed Edge and Cloud Computing	Enhanced Security	Accuracy my Degrade
[93]	Smart Health	Boosted Decision Tree using local ML model	Core ML with IPhone 8	Distributed Device and Cloud Computing	Enhanced Safety and Real-time detection	Further Validation is Required to Ensure Accurate Classification
[95]	Smart Health	SVM,KNN, Yolo3 and DLIB with ImageNet	keras library with UAV thermal and infrared cameras	Distributed Device and Cloud Computing	Detection with Reduced Response Time	Accuracy Could be Improved
[96]	Smart Health	ANN, CNN and RNN	Mobile Phone and sensors	On-Device Computing	High Accuracy and Real-Time Analysis	Does not Address IoT Device Diversity
[97]	Smart Health	Reinforcement Learning	Wearable devices and sensors	Edge Layer Computing	Improved Data Privacy	High Model Complexity
[78]	Smart Health	MobileNetV2 (CNN)	TensorFlow, Keras and OpenCV with NVIDIA Jetson Nano and Logitech USB camera C920e	On-Device Computing	Reduced Load to Cloud and Low System Cost	High Model Deployment Complexity
[98]	Smart Agriculture	LSTM	TensorFlow and Keras with Raspberry Pi 4 Model B	On-Device Computing	Load Reduction	Lower Accuracy
[81]	Smart Agriculture	ResNet-50 (CNN)	TensorFlow, Keras and OpenCV with NVIDIA Jetson Nano and Logitech WebCam	On-Device Computing	High Accuracy with Real-Time Detection	Does not Consider Data Diversity

Table A. - Evaluation of Neural Network Architectures

Script to train the Conv1D Classification Model without Native Bayes Classifier Compression on Dataset

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
file_path = "/teamspace/studios/this_studio/Documents/Individual Project/Datasets/merged_env_motion.csv"
df = pd.read_csv(file_path)
print("Columns in the dataset:", df.columns)
print(df.head())
df.drop_duplicates(inplace=True)
df.columns = df.columns.str.strip().str.lower()
numeric_cols = df.select_dtypes(include=np.number).columns
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())
columns_to_drop = ['smoke', 'device']
if 'co' in df.columns:
    columns_to_drop.append('co')
df.drop(columns=columns_to_drop, errors='ignore', inplace=True)
df['timestamp'] = pd.to_datetime(df['timestamp']).astype(int) / 10**9
if 'light' in df.columns and df['light'].dtype == bool:
    df['light'] = df['light'].astype(int)
target = 'motion'
if df[target].dtype == bool:
    df[target] = df[target].astype(int)
df.drop(columns=['hub', 'home'], inplace=True, errors='ignore')
features = [col for col in df.columns if col != target]
print(f"Features used: {features}")
print(f"Target variable: {target}")
scaler = StandardScaler()
df[features] = scaler.fit_transform(df[features])
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
X_all = df[features]
y_all = df[target]
classifier.fit(X_all, y_all)
```

```
sorted_nodes = np.argsort(classifier.theta_.sum(axis=0))
compression_percentage = 0.5
num_instances = int(len(df) * compression_percentage)
compressed_data = df.iloc[sorted_nodes[:num_instances]]
X = compressed_data[features].values.astype(np.float32)
y = compressed_data[target].values.astype(np.float32)
print(f"[Compressed] X shape: {X.shape}, y shape: {y.shape}")
TIME_STEPS = 5
BATCH_SIZE = 32
def create_windows(data, labels, window_size):
    X_windows = []
    y_labels = []
    for i in range(len(data) - window_size):
        X_windows.append(data[i:i+window_size])
        y_labels.append(labels[i+window_size])
    return np.array(X_windows), np.array(y_labels)
X_windowed, y_windowed = create_windows(X, y, TIME_STEPS)
print(f"Windowed data shape: {X_windowed.shape}, y shape: {y_windowed.shape}")
dataset = tf.data.Dataset.from_tensor_slices((X_windowed, y_windowed))
dataset = dataset.shuffle(1000).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
for batch in dataset.take(1):
    print(f"Batch input shape: {batch[0].shape}, Batch target shape: {batch[1].shape}")
    break
total_batches = len(list(dataset))
train_size = int(0.8 * total_batches)
train_ds = dataset.take(train_size)
test_ds = dataset.skip(train_size)
print(f"Number of batches in train_ds: {len(list(train_ds))}")
print(f"Number of batches in test_ds: {len(list(test_ds))}")
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(TIME_STEPS, len(features))),
    tf.keras.layers.Conv1D(64, 3, activation='leaky_relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling1D(2),
    tf.keras.layers.Conv1D(128, 3, activation='leaky_relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling1D(2),
    tf.keras.layers.Conv1D(256, 3, activation='leaky_relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
```

```
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.summary()
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)
history = model.fit(
    train_ds,
    epochs=20,
    validation_data=test_ds,
    callbacks=[early_stopping]
)
loss, accuracy = model.evaluate(test_ds)
print("Test Loss: {:.4f}".format(loss))
print("Test Accuracy: {:.4f}".format(accuracy))
model.save('conv1d_occupancy_model.h5')
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy')
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Model Loss')
plt.tight_layout()
```

```
plt.show()
```

Script to train the Conv1D Classification Model (with Bayes Naïve Classifier Compression)

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt

file_path = "/teamspace/studios/this_studio/Documents/Individual Project/Datasets/merged_env_motion.csv"
df = pd.read_csv(file_path)

print(df.head())

df.drop_duplicates(inplace=True)
df.columns = df.columns.str.strip().str.lower()

numeric_cols = df.select_dtypes(include=np.number).columns
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())

columns_to_drop = ['smoke', 'device']
if 'co' in df.columns:
    columns_to_drop.append('co')
df.drop(columns=columns_to_drop, errors='ignore', inplace=True)

df['timestamp'] = pd.to_datetime(df['timestamp']).astype(int) / 10**9
if 'light' in df.columns and df['light'].dtype == bool:
    df['light'] = df['light'].astype(int)

target = 'motion'
if df[target].dtype == bool:
    df[target] = df[target].astype(int)

df.drop(columns=['hub', 'home'], inplace=True, errors='ignore')
features = [col for col in df.columns if col != target]
print(f"Features used: {features}")
print(f"Target variable: {target}")
```

```
scaler = StandardScaler()
df[features] = scaler.fit_transform(df[features])

classifier = GaussianNB()
X_all = df[features]
y_all = df[target]
classifier.fit(X_all, y_all)

probs = classifier.predict_proba(X_all)
uncertainty = 1 - np.max(probs, axis=1)
sorted_indices = np.argsort(-uncertainty)

compression_percentage = 0.5
num_instances = int(len(df) * compression_percentage)
compressed_data = df.iloc[sorted_indices[:num_instances]]

X = compressed_data[features].values.astype(np.float32)
y = compressed_data[target].values.astype(np.float32)
print(f"[Compressed] X shape: {X.shape}, y shape: {y.shape}")



TIME_STEPS = 5
BATCH_SIZE = 32

def create_windows(data, labels, window_size):
    X_windows, y_labels = [], []
    for i in range(len(data) - window_size):
        X_windows.append(data[i:i+window_size])
        y_labels.append(labels[i+window_size])
    return np.array(X_windows), np.array(y_labels)

X_windowed, y_windowed = create_windows(X, y, TIME_STEPS)
print(f"Windowed data shape: {X_windowed.shape}, y shape: {y_windowed.shape}")

dataset = tf.data.Dataset.from_tensor_slices((X_windowed, y_windowed))
dataset = dataset.shuffle(1000).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)

for batch in dataset.take(1):
    print(f"Batch input shape: {batch[0].shape}, Batch target shape: {batch[1].shape}")
    break
```

```
total_batches = len(list(dataset))
train_size = int(0.8 * total_batches)
train_ds = dataset.take(train_size)
test_ds = dataset.skip(train_size)

print(f"Train batches: {len(list(train_ds))}")
print(f"Test batches: {len(list(test_ds))}")


model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(TIME_STEPS, len(features))),
    tf.keras.layers.Conv1D(64, 3, activation='leaky_relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling1D(2),
    tf.keras.layers.Conv1D(128, 3, activation='leaky_relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling1D(2),
    tf.keras.layers.Conv1D(256, 3, activation='leaky_relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.summary()

model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])

early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

history = model.fit(
    train_ds,
```

```
    epochs=20,
    validation_data=test_ds,
    callbacks=[early_stopping]
)

loss, accuracy = model.evaluate(test_ds)
print("Test Loss: {:.4f}".format(loss))
print("Test Accuracy: {:.4f}".format(accuracy))

model.save('conv1d_occupancy_model.h5')

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Model Loss')

plt.tight_layout()
plt.show()
```

Float16 Conversion from Float32 model

```
import tensorflow as tf

# Load trained Keras model
model = tf.keras.models.load_model("conv1d_occupancy_model.h5")

# Convert with float16 quantization
```

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types = [tf.float16]

# Convert the model
tflite_model = converter.convert()

# Save the model
with open("conv1d_occupancy_model_float16.tflite", "wb") as f:
    f.write(tflite_model)

print("TFLite model with float16 quantization saved.")
```