Agriscanner

A deployable weather station network and linked microclimate forecasting webapp

By

Adam Sidnell

Supervised by Professor Ruzanna Chitchyan

Department of Computer Science
UNIVERSITY OF BRISTOL

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering

September 2025

Word count: 15,941

Executive summary

This project report presents the design, development and deployment of a complete IoT weather station network for use in agricultural settings. The network was built from the ground up with off-the-shelf electronics and a custom weather proofed enclosure. The system streams one minute sensor measurements to a linked web application also developed in this project called Agriscanner (`https://agriscanner.onrender.com`).

The Agriscanner webapp provides users with historical, real-time, and forecast weather data through an intuitive and accessible interface. The webapp uses a machine learning model (LightGBM), to display microclimate specific forecast data that is trained by comparing general forecast information to sensor readings from the IoT network.

General forecasting models often miss smaller climate variations that exist in farms and fields. While commercial weather monitoring stations are often a trade-off between the range they can transmit information and their price. Because farms are remote, any device collecting information must be self-sustaining in power and able to send data over long distances.

The weather stations in this design achieved a measured range of 1,200m from the receiver in non-ideal circumstances and the final design is estimated to have over double the range of commercial options due to the inclusion of a repeater.

On the software side, the backend was built with a relational database that keeps a log of all sensor readings, along with a secure API that facilitates these database queries. It serves a front-end web interface that scored highly for usability on both mobile and desktop.

The machine learning model's forecast was compared to actual sensor data and showed that it was more accurate than the general forecast in most scenarios, but more training data is needed to improve its accuracy further.

It was an ambitious project for the timescale - involving the design and construction of four hardware devices; the development of a backend database and API; the development of frontend web interface and the training and deployment of a set of machine learning models.

Dedication and acknowledgments

I would like to thank my parents for the use of their garden, their wisdom and their near infinite patience; my supervisor for her advice and support throughout this project; and my partner, for helping me with range testing the network by walking the length of the Downs with a box of electronics, despite strange looks from passers-by.

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: Adam Sidnell DATE: 2nd September 2025

Table of contents

List of Figures

1 Introduction

In this report, I describe an online webapp called Agriscanner, which collects and displays microclimate data from an *Internet of things* (IoT) weather station network that I designed and built. The webapp can present live, historical and forecasted weather data to the user. The live and historical data is retrieved from a postgreSQL database that collects and stores readings from my weather station sensor nodes every minute. The historical readings were used to build a machine learning model that can predict future microclimate specific weather based on the general forecast.

The weather station network was built using off-the-shelf components and housed in hand made enclosures. The deployed system consists of four components: two field sensor nodes capable of reading weather data and transmitting it using a modern radio protocol called LoRa; a repeater that boosts received LoRa signals; and an internet connected gateway that uploads the weather data to an online database. The hardware in this project was tested in a private garden with deployment to Small Brook Farm in Exeter planned in future. Due to the timeframe this was not possible within the submission deadline.

1.1 Motivation and aims

Weather forecasts are typically based on data from distant weather stations and large scale models which fail to capture variations that exist within a farm or even a single field. These local variations, known as microclimates, can differ significantly even across relatively short distances due to differences in factors such as elevation, tree cover, or soil type.

For example, in a set of interviews with the owners of Small Brook Farm (Appendix B) the owners note that the use of traditional forecasts for their orchard is not particularly useful for them as weather forecasts tend to cover wide areas and are not always indicative of very local conditions. The owners mentioned that the weather conditions across a single field of apple trees was different, with higher winds on one side compared to the other.

This uncertainty regarding weather conditions has real world consequences for farmers. For instance, at Small Brook Farm wind speed is a key determinant of when the farmers can spray their crops with pesticides. Equally, general forecasts do not provide accurate enough information to enable farmers to predict a spring frost that damages crops or to assess the likely incubation period of pests and diseases.

Agriscanner aims to address this need for accessible local weather forecasting by giving farmers a weather station platform that can both relay current weather information from different sections of their field and predict future weather for these specific locations.

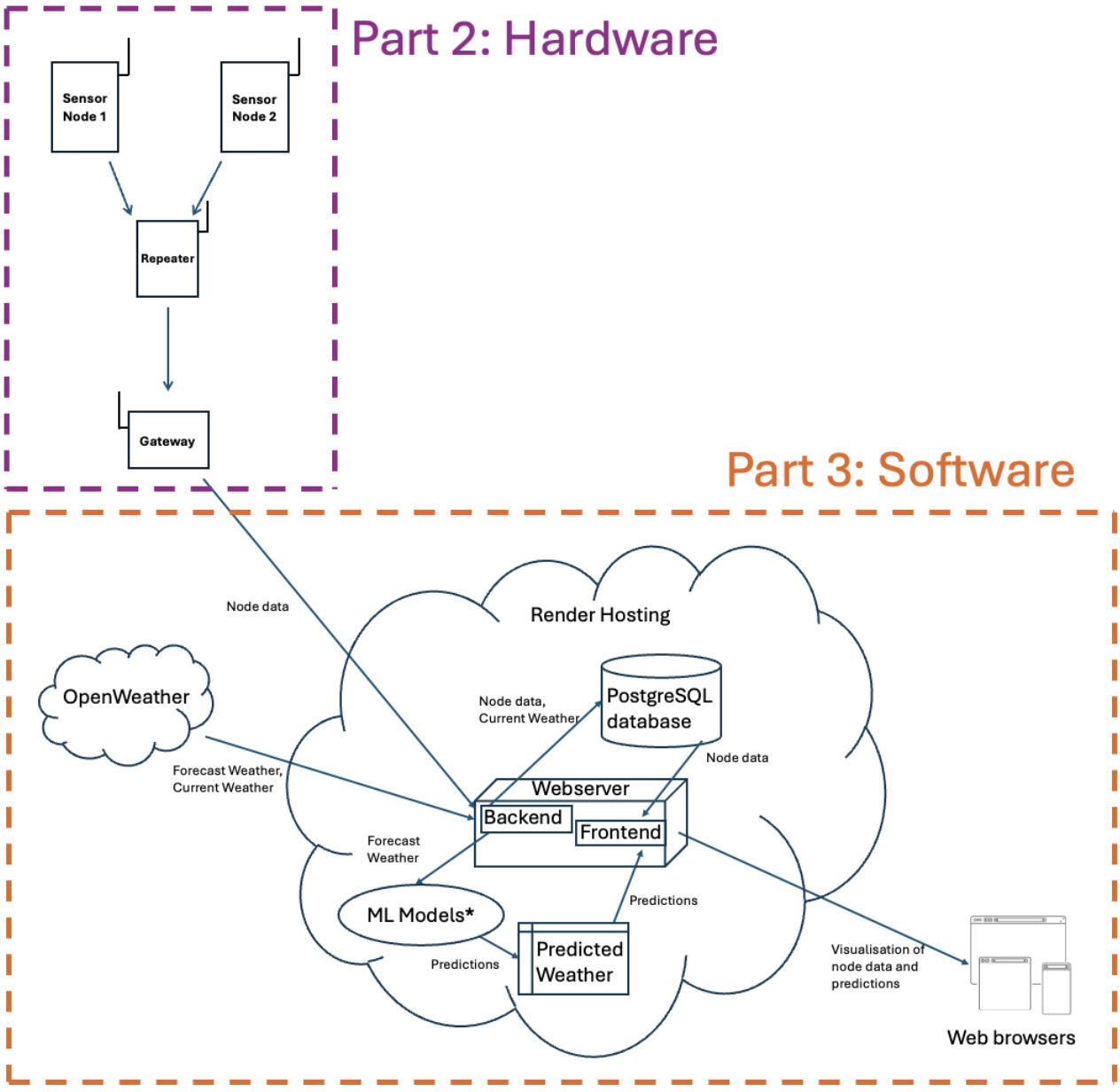
1.2 Contributions

The key contributions of this paper are:

- The development of a low cost, long-range remote weather station system with superior range and reduced cost compared to current solutions on the market. The long range allows for readings from the field to be communicated to the cloud without a need for internet access in the field.
- A publicly available web application for visualising live data from multiple field locations, that demonstrates a high degree of usability across both mobile and desktop platforms, as confirmed by quantitative System Usability Scale (SUS) evaluation.

- A method for enabling microclimate forecasting using a machine learning process that compares local sensor data with broader regional weather forecasts. This aims to give farmers access to a hyper-local forecast of future weather conditions that is specific to their field which helps them optimise agricultural tasks such as spraying.

1.3 Project schematic and layout



* Training the machine learning models is not pictured on this graph.

Figure 1: Schematic of entire project showing data paths

Figure 1 shows the design of the my entire project, which has been split into hardware and software categories that correspond to parts 2 and 3 of this dissertation. A zoomed in version of each category is shown on the overview of its respective part (see hardware overview in Chapter 5 , and software overview in Chapter 10).

Part 1 provides the technical background to understand the project, including an overview of the core technologies used and a review of related work in the field.

Part 4 then offers a critical evaluation of the system, discussing its performance, usability, and limitations, as well as highlighting opportunities for future development improvements and research.

Part I

Background

2 LoRa

Before analysing IoT systems more generally, I will explain the most critical hardware enabling technology for this project, which is the radio communication technique known as LoRa.

2.1 What is LoRa?

LoRa stands for **Lo**ng **Ra**nge and it is a radio modulation technique invented in 2014 that allows for the transmission of data over very long distances. It is one of several competing low power wide area networks (LPWAN) but the hardware to use it is much more available than these other networks. LoRa has a range over 4000 times greater than WiFi [1], a range that it can achieve with remarkably little power (Table 1). This makes LoRa the preferred technology choice for the remote, off-grid application in this project.

For a brief overview of the principles behind LoRa I have written the supplementary section "How LoRa works" in Appendix F. However, to summarise this, the most important principle that separates LoRa from traditional radio modulation is the fact that it can demodulate incoming signals more efficiently.

2.2 Benefits and limitations

The benefit from the ability of the LoRa receiver to demodulate signals more efficiently is two-fold. First it reduces the power needs on transmitter and receiver: the transmitter sends less powerful signals because the receiver can more easily distinguish signals; in turn the receiver can demodulate with a lower power budget because of the easier correlation process with LoRa chirps (Figure 46 from supplementary Appendix).

The second related benefit is the ability for the receiver to demodulate signals which are below the noise floor (the sum of all interfering signals). Essentially, even when background noise is 'louder' than the LoRa signal, the receiver is still able to distinguish and process the data in the signal. This helps LoRa transmitters to broadcast signals with a far greater effective range despite its low power. Table 1 summarises LoRa against other well known wireless communication technologies, showing the relative advantages or LoRa. The main disadvantage of LoRa is that it has a comparatively lower data throughput, however the packets involved in this project were only around 20-30 bytes so this was not a concern.

Technology	**Wireless Communication**	**Range**	**Tx Power**
Bluetooth	Short range	10 m	2.5 mW
WiFi	Short range	50 m	80 mW
3G/4G	Mobile network	5,000m	5000 mW
LoRa	LPWAN	2,000–15,000m	20 mW

Table 1: Comparison of wireless technologies (Source: [2])

3 *Internet of Things*

3.1 What is IoT?

The Internet of Things (IoT) refers to the concept of integrating networking capability in a range of devices, allowing for cooperation to reach common goals [3]. A similar but more colloquial term would be "smart" devices.

IoT has become increasingly common in recent years as a number of technological breakthroughs has made deploying these networks more practical. These advances, which are referred to as "enabling technologies", provide the foundation for modern IoT systems. LoRa is a key example described in Chapter 2, and a further list of enabling technologies is available as a supplement in Appendix G.

3.2 The layers of IoT

All IoT devices rely on the existence of three fundamental network layers [4] - working from the bottom to the top these are:

1. Perception - This layer contains sensors that collect information about conditions in the world around them. This layer may perform some transformations on the data it receives (e.g. sorting, formatting) or just transmit raw data. An example of this is a smart car charger collecting data on the charge level of an electric car.
2. Network - This layer acts as the bridge between the perception layer and the application layer. It is the medium and protocols associated with the transmission of data and the hardware necessary to interpret this. Following the above example this could be the use of WiFi or a mesh protocol such as Zigby to transmit the car's current charge level to a gateway - such as a WiFi router.
3. Application - This encompasses any software that manipulates, displays or otherwise uses data from the perception layer. This could be hosted on the cloud or locally. In the example above this could be an app that shows the current charge status of the car.

The hardware overview in Chapter 5 presents the components of my IoT solution in the context of these layers.

3.3 Studies using LoRa IoT weather stations in agriculture

The use of IoT in agriculture has become widespread in recent years as it offers the opportunity for farmers to improve yields and reduce costs by bringing digital solutions that would not have previously been viable in remote settings. The use of IoT in agriculture is often wrapped up in the moniker of "Smart Farming", which encompasses a range of digital, robotic, and internet-enabled approaches to improving efficiency. A 2019 review by Farooq et al. [5] reveals the scope of IoT applications in agriculture. These include precision farming (IoT weather monitoring), automated irrigation, pest and disease prediction with machine learning, and even the deployment of agricultural drones for spraying and imaging.

There have been a few studies around the use of LoRa in smart farming applications. In [6] LoRa was used in an edge computing exercise - a procedure to compress data from multiple LoRa sensor nodes is created and analysed. However, a criticism of this study is that they only implement a single sensor and all results are drawn from a computer generated sample. Additionally the range used for this single sensor to the gateway is only 200m, which is not a realistic distance for most agricultural purposes which the study is targeting.

The study in [7] implements a LoRa based weather station prototype in India. The authors create a fairly simple node with two sensor types. After this the data from the nodes is viewed using an OLED screen on the receiver. A significant criticism of this work is that it fails to specify the range at which the LoRa communication was tested, which is a critical parameter given that the technology was chosen specifically for its long-range capabilities. Additionally, while the paper suggests the model is suitable for large-scale deployment in a farm field , the experimental setup only consists of a single transmitter node and gives no details on whether it had any weather proofing.

This project aims to contribute to this literature by producing a more complete weather station prototype that is weatherproofed for outdoor deployment using multiple nodes and validated over more realistic distances.

3.4 Commercial IoT agricultural solutions with LoRa

While IoT weather station solutions are available commercially there are issues with using these.

Virtually all affordable weather stations rely on WiFi to stay connected. This is not viable for most farms as the fields they need to be deployed in are a long way from buildings with WiFi access. As discussed in section 2.2, WiFi has just a fraction of the range of LoRa so these types of stations are not comparable to the system I have developed. This is explored in greater depth in the evaluation (see Chapter 15).

4 Microclimates

This section gives some academic background on microclimates with a focus on their relevance to agriculture, and then looks at related attempts to predict microclimate weather.

4.1 What is a microclimate?

A microclimate is generally understood as a set of distinct climatic conditions within a small, localised area [8]. The maximum size of a microclimate is debated, but the World Meteorological Organisation (WMO) regards it as occupying an area of anywhere from less than one metre across to several hundred metres [9]. In practice, microclimates can occur in spaces such as gardens, valleys, caves, or fields. Even human-made structures can generate their own microclimates; for example, tall buildings can create *street valleys* that reduce wind flow and lead to the formation of localised pockets of warmer air, which can trap higher concentrations of pollution from vehicle emissions [10]. Vegetation plays a critical role in influencing microclimates. The addition of trees to an urban environment can reduce air temperature by as much as 2.8 °C [11].

4.2 Microclimates in agriculture

Microclimatic variations have profound implications for agriculture, as the climate that crops are exposed to has an enormous impact on overall agricultural yields. Indeed, farmers have modified the microclimate of crop fields for millennia, a clear example of this being the use of fencing to reduce soil erosion and damage to edible plants [12]. Therefore, the relationship between microclimates and agriculture has been the subject of extensive research, particularly as climate change introduces new threats to food security.

A Danish study by Haider et al. investigated how agricultural pests and diseases are influenced by microclimatic conditions. Temperature sensors were installed in six different areas with distinct microclimates (such as hedges and cattle fields). The data revealed that the daytime temperature in these microclimates was significantly higher than those predicted by Danish weather forecasts. Using these measurements, the researchers then estimated the incubation periods of various pests

and diseases, demonstrating that elevated temperatures in microclimates could shorten incubation times and thus accelerate the risk of disease outbreaks [13].

Another important aspect of microclimates for farmers is how it can affect frost risk - sudden and unpredictable drops below freezing in crop fields that damage plants and are particularly common in spring. A principal factor for this is a lack of "cold-air drainage". In areas with depressed topography, cold, dense air can accumulate to form pockets where temperatures can be several degrees lower than the surrounding landscape [14]. These local conditions are not captured by general weather forecasts highlighting the need for more precise monitoring and development of effective early warning systems.

4.3 Microclimate prediction using machine learning

There have been a number of recent studies where machine learning has been used to help predict micro climate conditions in agricultural settings. General weather models operate at magnitudes between 1 and 10 km and microclimate predictions require models that operate at scales of roughly 100m or less. Using current numerical weather prediction models for micro-scale predictions is computationally expensive [15], and these models have lower accuracy rates than predictions using machine learning due to the inherent complexity and non-linear nature of microclimates.

A number of the studies described here focus on comparing the accuracy achieved by different types of machine learning, and highlighted a distinction between machine learning (ML) and deep learning (DL). They are both systems that learn from data and make decisions based on datasets provided when the systems are trained. ML techniques include decision trees, regression and neural networks. DL systems are a type of ML that uses neural networks with many layers and is used to model more complex patterns with larger datasets. The table below provides a summary of the studies included in this section

Table 2: Selected summary of machine learning studies

Study	Location	Type of learning	Learning framework	Source of forecast data	Source of training target data
Agriscanner (This work)	Outside rural - South Gloucestershire, UK	ML	LightGBM	OpenWeather	Sensor readings
Kumar et al. (2021)[16]	Various	DL	LSTM	Not specified	Sensor readings
Zanchi et al. (2023)[17]	Outside rural; Italy, Lombardy	DL	Feed-forward neural network	ERA5 ARPA	Sensor readings
Blunn et al. (2024)[15]	Outside urban; London, UK	ML & DL	Multilayer Perceptron Random Forest XGBoost	UKV weather data	Citizen weather stations
Abdelmadjid (2025)[18]	Greenhouse - Unknown	ML & DL	CNN-LSTM LSTM (DL) SVM-RBF Prophet LightGBM XGBoost	Public kaggle dataset	Public kaggle dataset

A 2021 study by Kumar et al [16] developed an ML framework called DeepMC as a part of a Microsoft Research initiative. Their model is able to predict a variety of climatic variables such as soil moisture, wind speed and temperature using inputs from weather station forecasts and IoT sensors. They were able to get up to 90% accuracy with a 12-120 hour forecast range.

Zanchi et al [17] used physical modelling of local terrain combined with DL to forecast the microclimate in the foothills of Lombardy. The objective was to predict the local conditions at the meter-scale as opposed to the 10km+ scale of regional and global weather forecasts. The initial model combined data about the morphology of the local terrain and weather forecast data to provide the input data for two feed-forward neural networks. These neural networks were trained to predict the local weather variables using data from 25 sensors deployed in the region being studied. The study demonstrated that local predictions were more accurate when using forecast data from local weather stations (ARPA) as opposed to global climate datasets (ERA5), but accuracy was still good either.

This study is of particular interest for this dissertation as it used IoT sensors to generate the training targets for the neural networks and in this regard is the most similar to my design. The paper also includes a discussion of problems with reliability of the sensors - only 4 out of 25 units ran without failure over the period of the study - and the need to clean the data to ensure these failures were excluded from the training data. In my project, I came to appreciate how time consuming the development of hardware is, and how many unexpected issues can occur. As this project had a hard end date, this affected the quantity of data I was able to collect and therefore ultimately impacted the training of the machine learning models.

Blunn et al [15] ran a study focussed on predicting temperatures in urban environments during heatwaves, using data from eight heatwaves in London, UK. They used data from the UKV - a high-resolution weather forecasting model - and from citizen weather stations (CWS), made available via the Weather Observations Website - a cloud platform where individuals can upload data from their personal weather stations. The study tested a variety of ML models including one DL model. The authors used a similar model training design to that used in this dissertation. The models were trained on UKV variables (i.e. a general forecast as with mine) and CWS variables (i.e. sensor information) to bias correct the UKV readings and create a forecast prediction model that could predict the CWS readings accurately (mean average error: 0.12°C) compared with the general weather readings from UKV (mean average error: 0.64°C). The main point of difference to my own study here is that I am using four different climate variables compared to just temperature in the Blunn paper. Also CWS is publically available civilian weather stations whereas I am using my own sensor data from a devices I have built.

A very recent paper from Abdelmadjid et al 2025 [18] used online datasets from Kaggle to develop an ML tool to predict changes in temperature and humidity within greenhouses in response to changes to external weather conditions. They used this data to test three ML models and three DL models and selected the LightGBM ML model and the LSTM DL model as the best performing models for prediction. The overall system design consisted of four LSTM models feeding into the LightGBM model. This design resulted in 98.45% accuracy for temperature predictions and 99.61% accuracy for humidity predictions. This study informed my decision to use the LightGBM model for my system as they found it to be reliable and effective in capturing underlying data patterns.

Part II

Hardware development

5 Overview of hardware

The design of the IoT system involved three distinct module types: the sensor nodes and repeater which are located outside and the gateway which is located inside a building with WiFi access. The system consists of two separate sensor nodes that are placed in range of the repeater. The repeater is then placed in range of the gateway endpoint to allow the uploading of climate data to the cloud.

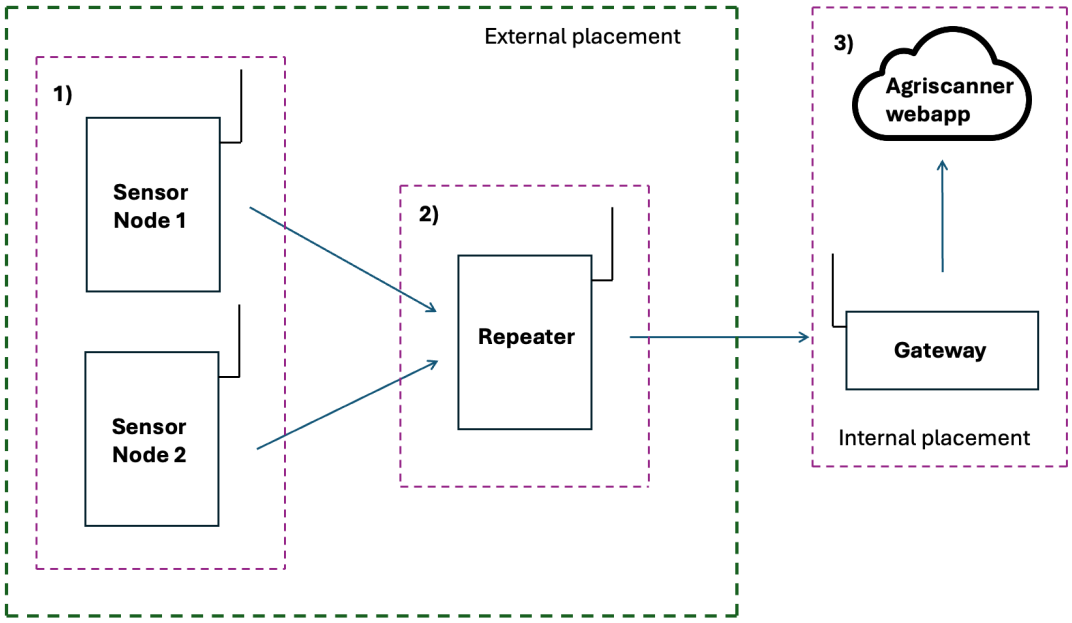


Figure 2: Network diagram of the system

1. Sensor Node: The two nodes in this part of the network are in the perception layer. The nodes collect readings on temperature, humidity, wind speed and soil moisture levels. The results are collated into a comma separated string which is then emitted as a single packet from the LoRa transmitter.
2. Repeater: This module is part of the network layer of the system as it facilitates communication between the perception layer and the application layer. The addition of a repeater node effectively doubles the range of the system. The repeater reads and decodes received LoRa signals from the sensor nodes. It then adds signal strength information to the string and re-emits the LoRa signal. The repeater has no sensors but is otherwise identical to the sensor nodes.
3. Gateway: The final part of the hardware system is the gateway - which is also part of the network layer. This module has mains power and a WiFi connection. The gateway receives LoRa signals from the receiver and uploads the data to the cloud.

6 Design

6.1 Node Components

As the sensor nodes and the repeater are situated outside and rely on solar power, they required components that were highly power-efficient, while also being capable of transmitting small data packets via LoRa. An equally important consideration for the design was how to weatherproof the final enclosure to protect the sensitive electronics from water ingress and environmental damage.

6.1.1 Challenger RP2040 LoRa

The iLabs Challenger RP2040 LoRa (Figure 3)is an embedded computer that uses the Raspberry Pi RP2040 chip that was released in 2021. The RP2040 itself is a low-cost and power-efficient processor easily capable of performing the data encoding and transmission in my use case. Additionally, the chip is extremely popular with over 10 million units being produced in the first two years of release [19]. This popularity means there is ample documentation for developing with this processor.

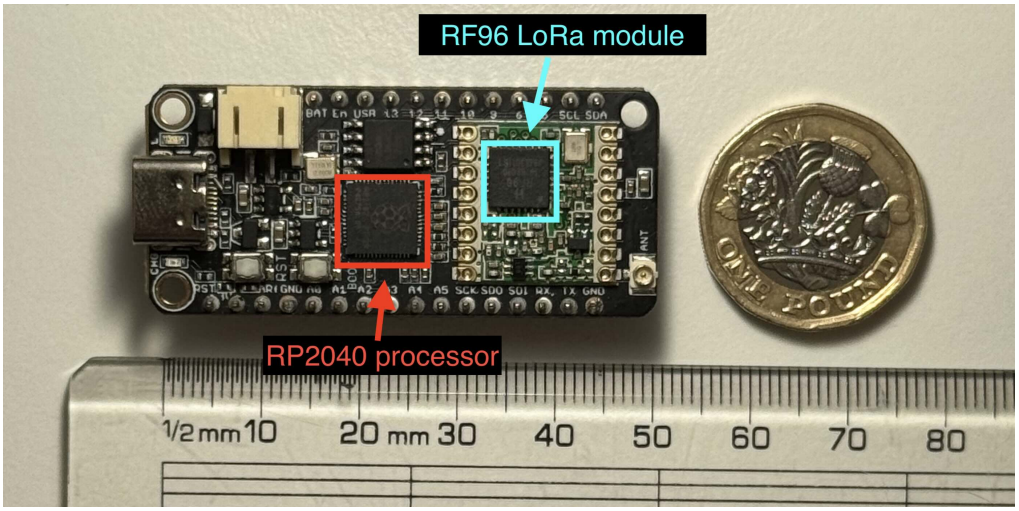


Figure 3: iLabs Challenger RP2040

The Challenger board itself is well suited for this project for several reasons. It uses the compact Adafruit feather form factor, giving a board dimension of just 5cm by 2cm, making it easy to mount in a small enclosure. The onboard Hope RF96 LoRa modem is built directly into the board and the U.FL antenna connector allows for the swapping of antenna's to different varieties. This board's LoRa module is also set to transmit at a frequency of 868mhz which is a standard UK frequency for LoRa and gives a good balance between range and bandwidth.

Another useful aspect of the board is the abundance of GPIO pins (20 in total) allowing for a large number of sensors to be fitted to the board.

6.1.2 Antennae

The selection of antennae is one of the largest determinants of range and reliability in the context of wireless communication systems [20]. Initially I used a simple PCB antenna as shown in Figure 4, however as explained in Section 7.1, the range of this was insufficient for my use.

Figure 4: Low range PCB antenna

To improve overall range I switched to a more capable omnidirectional whip antenna that was made specifically for the Challenger RP2040. The antenna is tuned to perform best at the 868mhz frequency range - which is the range I was using.

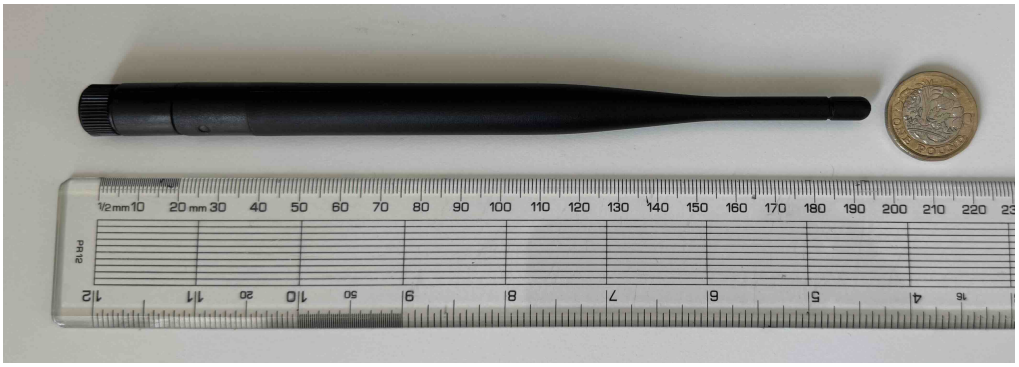


Figure 5: iLabs whip style LoRa antenna (868mhz)

6.1.3 Powering the node

To allow for continuous operation away from power sources, I attached a 6W Monocrystalline Silicon Solar Panel (Figure 6)to each of the nodes. I also installed a solar power management module (Figure 6) onto the Challengers. This module moderates the output from the solar panels which is at too high a voltage to directly power the nodes and repeater. I also incorporated a battery (not pictured) that charges from the solar panel output and provides power to the microcontroller when solar power is not available.

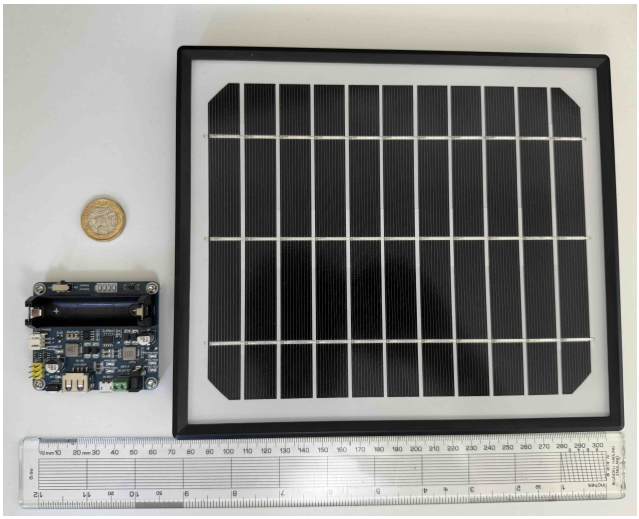


Figure 6: Waveshare solar power management module (left), solar panel (right)

6.1.4 Sensor selection

Temperature and humidity sensor

I used a DHT11 temperature/humidity sensor (Figure 7) for each sensor node to provide basic readings. It was chosen for it's low cost, availability and compatibility with both the RP2040 Challenger and CircuitPython (via libraries). The sensor can be connected to the microcontroller using a single GPIO pin as well as the usual power and ground pin.

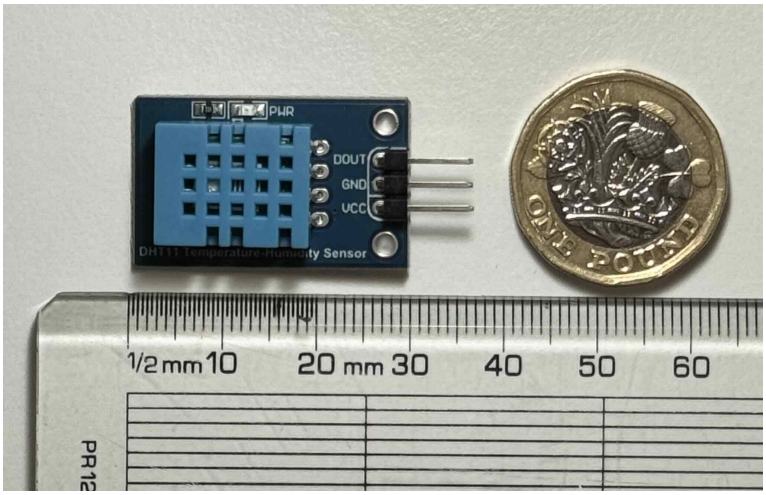


Figure 7: Waveshare DHT11 temperature/humidity sensor

Soil moisture sensor

The main choice between soil moisture sensors is whether to use a capacitative or resistive style sensor. In a resistance sensor the diodes must be bare plated metal in order for resistance between each diode to be measured. However the disadvantage of this is that bare metal corrodes in the presence of water, and corrosion affects the accuracy of the sensor. The benefit of capacitative sensors is that the diodes are covered by a protective layer making them much less susceptible to corrosion. For this reason I chose a capacitive sensor (Figure 8).

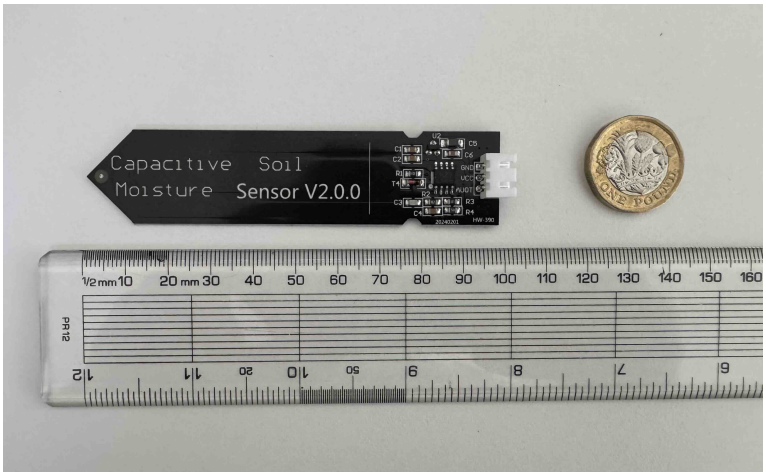


Figure 8: The Pi Hut capacitive soil moisture sensor

Wind speed sensor (Anemometer)

The anemometer chosen (Figure 9) was the DFROBOT wind speed sensor. It balances value with construction quality as unlike many other cup style sensors this is made of metal and rated for outdoor use. When I chose this component I was unaware that the RS485 serial communication protocol that the sensor used was not compatible with the Challengers UART protocol. Fortunately, I was able to purchase an RS485 to UART serial converter that allowed the devices to communicate.

Figure 9: DFROBOT wind speed sensor

A second issue was that the anemometer needed a 7V-24V input voltage to take readings while the maximum voltage the Challenger can provide was 5V. To remedy this I bought a 9V step up converter to boost the Challenger's voltage to the required voltage range.

6.2 Gateway

The components for the gateway consisted of a Challenger RP2040 (with whip antenna) connected to a Raspberry Pi (Figure 10). This Raspberry Pi was significantly more powerful than was needed but it was already available from the University for use in this project. The Challenger first receives and decodes messages from the repeater. The Raspberry Pi reads the decoded LoRa message from the serial output and then sends a POST response to the backend API which inserts sensor data into a SQL database.

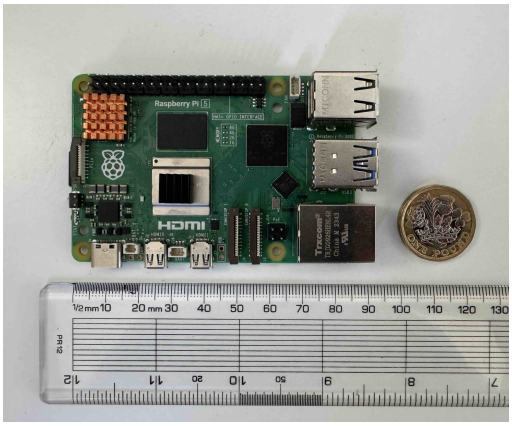


Figure 10: Raspberry Pi 5 (8GB)

7 Development and testing

7.1 Range tests

One of the most important tasks to carry out prior to deployment in the field was testing the range of the devices.

For this experiment I took four Challenger RP2040s to The Downs, a large public park in Bristol. Here I tested four different antenna configurations to compare how well the signal travelled across an increasing distance. Signal strength can be measured using the received signal strength index (RSSI), a measure of the difference in signal from the transmitter to the receiver that uses decibel-milliwatts as a unit (dBm).

For the test, two Challengers were programmed as transmitters, sending an example data packet similar to the data that would be generated by the sensor nodes. One of the transmitters used a simple PCB antenna (Figure 4) while the other used a higher range whip style antenna (Figure 5). Then I programmed the remaining two Challengers as receivers with one using a low range antenna and the other a long range one.

This meant that four different antenna configurations could be tested concurrently, as the receivers could pick up the signal from each transmitter. Before performing the test I estimated that the maximum range of each configuration would look like the below:

1. Whip antenna to whip antenna (Likely best result)
2. PCB antenna to whip antenna
3. Whip antenna to PCB antenna
4. PCB antenna to PCB antenna (Likely poorest result)

The reason I predicted the PCB transmitter to whip receiver would outperform the whip to PCB configuration is that improving receiver sensitivity (i.e. the ability of the receiver to 'hear') is more efficient than increasing the transmitters effective radiated power (how loud it shouts) [21].

Nine different distances from transmitter to receiver were tested, in 200m increments starting from 0m as a baseline to a distance of 1600m (Figure 11). An important aspect in getting a successful LoRa connection is whether there is line of sight between the transmitter and receiver. Figure 12 shows the elevation profile of the test area, with the initial large dip being an inaccuracy from Google Earth's topology data as the 0m point is near a cliff edge. The lowest elevation point is at the starting point at around 83m above sea level, while the highest point was at the 1km mark at around 94m making an elevation range of 11m. My hypothesis was that signal would likely drop off or stop entirely beyond this point as points beyond 1000m would be below the hill line. This would effectively mean that the receivers would be in a signal shadow point where the transmission waves would not be able to reach them. The only possibility for signal to reach this area would be from reflections either from nearby buildings or topology. The effects of reflections are virtually impossible to account for in the real world and therefore no accurate predictions could be made.

Figure 11: Google earth image of data collection points

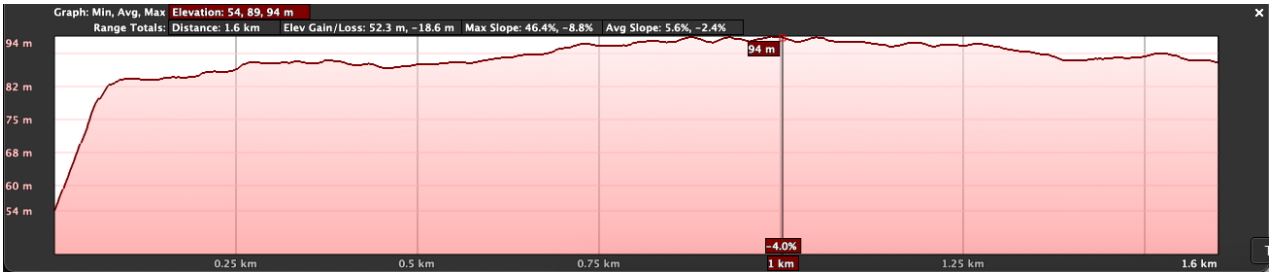


Figure 12: Elevation profile of test area (ignore large dip at start)

Unfortunately, at the time of the test a festival was being run between the 1200m and 1400m mark. The festival had a number of large tents and temporary buildings constructed which very likely negatively affected the signal. Final data for the range test is shown below.

Figure 13: Graph to show signal loss from different receiver and transmitter configurations (higher is better)

Unsurprisingly, the results show that the whip to whip antennas had the best performance. It was the only configuration that consistently received signal all the way to 1200m. In contrast with my prediction, the whip to PCB configuration was the second best and even received a signal at 1200m; although this was less consistent and had a lower signal strength compared to the whip-whip. I am unsure why this was superior to the PCB-whip but clearly the increased effective radiative power from the transmitter had a greater bearing on the RSSI than the sensitivity at the receiver end. The PCB-whip managed only half the range at 600m and the PCB to PCB managed just 200m, demonstrating its inadequacy for application in my project.

Additionally, the graph shows that LoRa signals appear to not be interpretable below an RSSI of -130dBm. The whip to whip configuration only went down to -120dBm, suggesting there was still scope to increase range further. These results are discussed further in Section 15.1.3.

7.2 Battery tests

There is no mains electricity available in the apple field so the nodes must be able to operate without an external power source. Even with the use of a solar panel the node must be able to work through the night and during periods of dense cloud coverage where the solar panel will not have sufficient power to keep the node running. This is where the use of a battery is essential as the solar panel can charge the battery with excess energy during periods of excess solar radiation, such as during midday hours, and then store this energy for periods of low or no solar radiation.

However, while daylight will be of little concern during the summer months when this dissertation is being written, there will be far fewer days of usable sunlight over the winter period. As the UK has very high latitude, there is large seasonal variation in the length of a day. For the town of Crediton (the nearest town to Small Brook Farm), in the summer there are 16.5 hours of daylight while in

winter it receives only 7.6 hours; making one night 16.4 hours long. Therefore the node must have a battery sufficiently large to power it for a minimum of 16.4 hours with some additional charge to account for high cloud cover during the early evening and morning period.

To see if the battery solution was sufficient for this environment a test was run on a fully charged 18650 battery that was then connected to the solar power manager to allow the voltage to be regulated up to the Challenger's 5V input voltage.

A digital multimeter was installed between the solar power manager and the Challenger's USB C input to view the voltage and current in real time as well as running a timer for the test and a calculation of the number of watt-hours consumed by the node. The node was then run with a full suite of sensors attached. During the test the node used 80ma at 5V for a wattage of 0.4W.

The node ran until the battery would no longer discharge, with the final battery life of the node being 19 hours and 17 minutes. The meter showed that the node consumed 7.96Wh of power. Considering the battery capacity is 9 Wh it may seem that the battery ran out too quickly. However, the solar power manager documentation [22] tells us that the battery boost efficiency - being the conversion of battery voltage from native 3.7V to 5V output - is only 86%. With that in mind the predicted effective capacity is 7.74Wh, which is very similar to the actual result.

The achieved result of 19 hours should be sufficient for running the node continuously for much of the year as this compares to the longest night period of 16.4 hours. However, during periods of heavy cloud cover in winter there is a chance the node would not be able to charge the battery sufficiently in the day to allow the node to run over night. Later on in development, I added an additional battery in order to improve battery life further (Section 8.1.1).

7.3 Solar panel testing

The solar panel is rated for 6W maximum power. This however would only realistically be achieved under optimal conditions with a clear sky and full sun around midday. This power rating was verified using a multimeter on such a day where a voltage of 7.02V and an amperage of 0.87A was measured, giving a final power of 6.1W.

As explained in the battery section above, the node consumed about 7.96Wh over a 19 hour and 17 minute period. We can therefore determine that the node requires roughly 10Wh of energy per 24 hours to able to run continuously. It would be tempting to then say that the node would need well under 2 hours of ideal sunlight to operate for an entire day (being 6W * 2h = 12Wh). However we must also account for the energy loss when converting from raw solar output to the regulated 5V input that Challenger accepts.

The solar panel manager rates its conversion efficiency for solar at 78% meaning for each watt of solar energy received 22% of this will be wasted as heat when converted to the correct voltage. If we adjust for this loss we would need effectively 12.8Wh of energy just to power the node for a day, which equates to 2 hours and 8 minutes of ideal conditions.

An additional 9Wh is needed to charge the battery which if we adjust again for solar efficiency factor we get 11.5Wh. This is an additional 1h:55m of ideal sun, meaning in order to charge the battery from 0% to 100%[1] and power the node for a whole day requires of roughly 4h:03m of sunlight.

7.4 Hardware assembly and weatherproofing

Since most of the hardware in this project contains exposed electronics, the final assembled devices needed to be made wind and water resistant to prevent failure. However, the sensors and solar panel also needed to be exposed to perform their function effectively - e.g. a solar panel must have a clear view of the sun throughout the day. The final design therefore needed to balance multiple conflicting purposes:

[1] It is unlikely that the battery will fully deplete overnight, so this is a worst case scenario

- The core electronics (Challenger, solar manager, battery etc) must be kept dry, cool and away from wind which may damage electronics.
- The wind sensor must be exposed to the elements and securely fastened to withstand intense wind, it must also be kept high off the ground for more accurate readings.
- The solar panel must be exposed to the elements and be south facing at at an angle of around 40 degrees to optimise solar efficiency [23].
- The soil moisture sensor must be inserted into the ground and as it has exposed electronics must be made waterproof.
- The antenna of the Challenger must be placed as high as possible (at least 1.5m from the ground).
- The temperature/humidity sensor must be exposed to the elements to ensure accurate readings but cannot be exposed to rain due to the exposed electronics on it.

Due to the conflicting nature of some of these requirements, the final device would need to allow for positioning of different components at different heights.

The final design therefore was to place most of the sensitive electronics inside a waterproof box with cut-outs for wiring for external components. This was then mounted onto a 2m pole to allow for a good antenna signal and more accurate wind speed readings.

7.4.1 Sensitive electronics

An IP65 waterproof junction box (Figure 14) was selected to house the non-sensor portion of the electronics. An IP rating measures the Ingress Protection of a devices housing against water and dust defined by the International Electrotechnical Commission. The first number of an IP code is a measure of dust resistance while the second number is a measure of water resistance. An IP code of 65 therefore means the box used here has perfect dust resistance and can withstand water jets being sprayed at it from multiple directions [24]. This level of protection should be more than adequate to withstand the heavy rainfall it will experience.

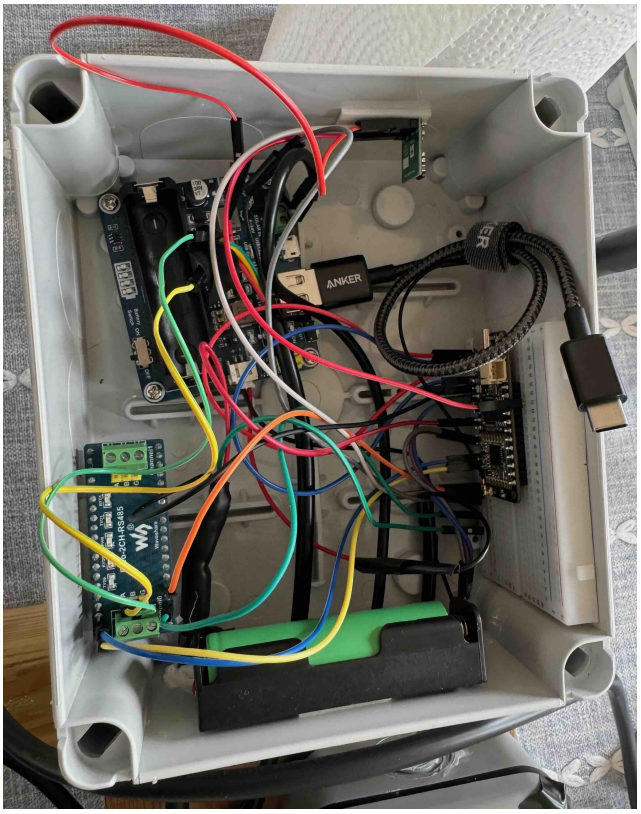


Figure 14: Open junction box showing internal wiring

The Challenger microcontroller was inserted onto a half size breadboard and then stuck to one side of the box. The antenna cable was routed to the top right corner where a small hole was drilled to allow for the antenna to be mounted externally. All the other components were then wired into the breadboard and stuck down with double sided sticky pads to prevent damage during transport or heavy winds. The bottom of the box had holes drilled for the anemometer, soil moisture sensor, solar panel and temperature/humidity sensor respectively. To prevent water ingress into these holes, sealant was deposited over the holes.

The entire box was attached to a plank of wood measuring roughly 25cm by 60cm. This board holds all the sensors with the exception of the soil moisture sensor which must be able to hang freely.

7.4.2 Soil moisture sensor

The sensor that required the most thought in respect of waterproofing and mounting was the capacitative moisture sensor. The first challenge was the need for the soil moisture sensor to be able to reach ground level while the microcontroller continued collecting its readings 2m above ground level. The solution to this was to swap the small included wires with a 2.5m weatherproof cable containing three cores (one for power, one for ground and one for data). Each cable core was then soldered onto the sensor as in figure 15, allowing for the sensor to be inserted into the ground.

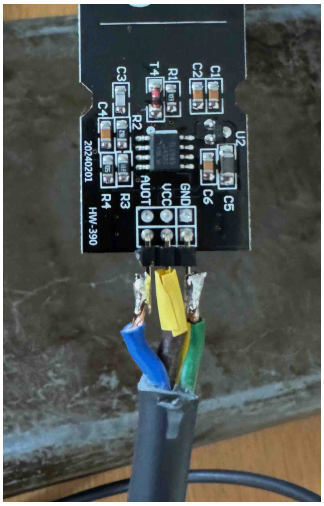


Figure 15: Soldered soil moisture sensor

Waterproofing the sensor was necessary as while the bottom three quarters of the board are designed to be inserted into soil, the part above this is made up of exposed electronics that must not come into contact with water (Refer to figure 8). This means it was essential to waterproof this part of the board.

To achieve this I followed an online tutorial on a hobbyist website [25]. and painted the electronics using nail varnish. While this sounds unconventional, nail varnish is a non-conductive compound that can be easily painted over electronics using the brush included and so is quite a popular low-cost method for waterproofing electronics. After this is applied the portion of the board with the electronics is covered in heatshrink to form a tight seal over the board, with a layer of nail polish on the edges to reduces the chance of water ingress under the heat shrink's plastic.

7.4.3 Other external components

The DHT11 sensor was mounted inside a smaller IP55-rated junction box, which provides the same resistance against rain as the larger box. To allow for more accurate readings, small holes were drilled on the bottom panel of this so the air temperature inside the box would better match the external temperature/humidity. To further improve the sensor's accuracy, the box was painted white to reflect sunlight, and a solar shield made from a cut and reshaped aluminium can was mounted to the south facing side. This shield (see red box in Figure 15) helps to prevent the sensor being exposed to direct sunlight, reducing temperature distortion while still allowing airflow from the sides.

The solar panel was mounted below the main enclosure using the included adjustable gimbal mount. It was fixed at an angle of approximately 40° from horizontal, which balances solar efficiency throughout the year [23]. This angle is steeper than the optimal summer setting but allows for improved performance during winter months when the sun is lower in the sky. The gimble allows for rotation in all planes which is useful in case the box itself cannot be mounted in a perfect southerly direction.

The anemometer was secured to a separate length of wood, positioned approximately 0.5 metres away from the main unit. This separation reduces turbulence caused by the box and pole, leading to more accurate wind speed readings. The sensor was screwed directly into the wood and can be positioned at a height of 2m (although in test deployment they were mounted lower for easy access.).

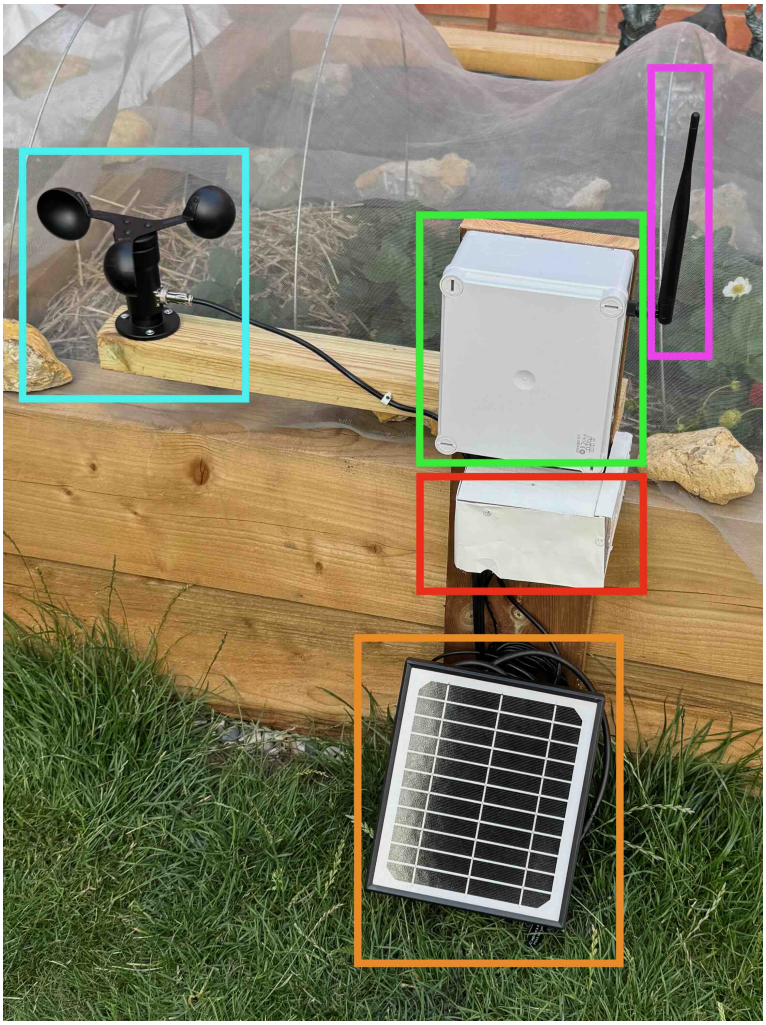


Figure 16: Final node design

Figure 16 shows the final assembled node. From top to bottom, the purple box shows the antenna, the blue box shows the anemometer on its separate arm, the green box contains the electronics, the red box is the solar shield that surrounds the temperature/humidity sensor and the orange box shows the solar panel. Not pictured is the 2m wooden post that extends the height of the node and the soil moisture sensor which is inserted into the ground behind the solar panel. The repeater looks similar and just lacks the anemometer and solar shield in this image.

7.4.4 Gateway

Figure 17: Completed gateway

The gateway node (Figure 17) required less careful engineering as there was no requirement for it to be waterproofed or mounted externally. As this would eventually be placed inside a private home, the aim was to make it minimally imposing so a design similar to an internet router was chosen.

The two devices inside the box were the Raspberry Pi and the Challenger. A plastic box was used with cut outs for the Pi's outputs and power cable, with an additional cut out on the lid for the Challenger's antenna. The Pi and Challenger were then secured to the box's base with velcro tape, as both devices would potentially need to be removed for trouble shooting.

8 Deployment

Before setting up the nodes at the farm it was important to run the equipment in an area where repairs and updates could be more easily performed. The nodes were therefore placed at different locations in a private garden and ran continuously from the 15th of August.

8.1 Challenges and solutions from test deployment

8.1.1 Battery capacity

An initial challenge identified during the test deployment was that the battery capacity on the sensor nodes was insufficient for continuous 24/7 operation. This became evident on the fourth day of the deployment when, following an overcast evening, one node ceased transmitting at approximately 6:00 AM due to power depletion.

To resolve this, the battery capacity on each sensor node was doubled by adding a second battery. The repeater node did not require this upgrade due to its lower power consumption. Post-upgrade, monitoring of the battery voltage confirmed that the existing solar panels were sufficient to fully recharge both batteries by early afternoon each day. This modification proved largely effective, although there were two more losses of power during the night even after the upgrade. Performance of batteries is explored more in Section 15.1.1.

8.1.2 Behaviour on loss of power

A related problem was that when the node died it did not start sending LoRa packets again even after power was restored. I discovered that when the microcontroller detected a brownout - a sudden

dip in voltage - the device would enter a safeboot mode. In this safeboot mode the default code.py file would not automatically be run and instead the device would create and use a safemode.py file that would essentially run indefinitely unless the reset button on the device was pressed or the device was powered off and on again.

Clearly this behaviour was not appropriate for field deployment where brownouts would potentially occur whenever the battery was discharged completely. Even with the additional battery capacity provided by a second battery there would still be a high chance of this occurring on particularly dark days during winter, where continuous uptime could not be guaranteed.

To remedy this, I modified the safemode.py file using a template I found in the CircuitPython documentation [26]. This version of the safemode.py file was made specifically for the case I was using i.e. remote solar powered projects where a manual reset of the board could not be achieved easily. Instead of entering an infinite loop this safemode is designed to enter a low power mode for a short period and then try to cycle power back on.

Disappointingly, the addition of this new safemode was not reliable and occasionally the device would enter an unknown state where a hard manual reset was necessary. This problem is discussed further in Section 15.1.4.

8.1.3 Temperature-humidity sensor behaviour in sun

A separate issue was the behaviour of the temperature-humidity sensor when exposed to direct sunlight. While the junction box used to encase the sensor was painted white to reduce any heat transfer from solar radiation, the readings taken on very sunny days showed a large delta of around 5 celsius between the sensor readings and local air temperature readings that could not be explained with the existence of a microclimate.

To reduce this effect an additional solar shield consisted of a wooden frame and aluminium shield was made. This was positioned to surround the sensor to block light falling directly on the junction box. This appeared to fix the issue and readings were more in line with the expected amount.

Part III

Software development

9 Programming the hardware

The Challenger boards can be programmed in a few different languages. The most common of these are C (including C++) and Python. I decided to program the boards in Python due to the large availability of sensor libraries written in this language. Programming in C, while potentially more efficient, would likely have meant I would need to write my own libraries and functions to get many of the sensors to function properly and therefore was not selected.

The RP2040 processor of the Challenger uses relatively low power, so the versions of Python for the chip tend to be stripped down to accommodate this. The two main Python versions available are CircuitPython and MicroPython. They both have extensive libraries and work with all the sensors I have selected. With no major difference in their functionality, I settled on CircuitPython as this was the version recommended by iLabs (the makers of the Challenger board) and included a library that supports the Challenger as well as example code. As previously mentioned and explored further in

the evaluation (Section 15.1.4), CircuitPython seems to have a dysfunctional reset behaviour when power drops out, so on reflection perhaps Micropython should have been used instead.

The development process for the sensor nodes, repeaters, and gateway was similar, so the rest of this section will briefly summarise how the sensor nodes were programmed for, as they presented the greatest challenge. The full code for sensor node 1 is available in Appendix H.

I began with the example code that comes as standard on each of the Challengers [27], this was a simple "ping/pong" test program but the LoRa radio initialisation code was very helpful here. I then adapted this program for the sensors outlined in Section 6.1.4, adding the necessary Python libraries for each component. Acquiring readings was relatively straightforward for all sensors except the anemometer.

The anemometer used a communication protocol not natively supported by CircuitPython, which meant I had to build the function `get_wind_speed()` based on the developer's documentation [28] to acquire readings. Getting this function to work with the wind speed sensor was quite time consuming and this software issue was in addition to needing two additional modules to power and connect the device (Section 6.1.4). This made the anemometer one of the most difficult hardware and software challenges of the project.

Once this issue was resolved, the readings were converted to CSV format, and the sensor nodes began transmitting successfully.

9.1 LoRa settings configuration

The most fundamental part of the program for each of the nodes in the LoRa network was that they could communicate with each other. This required the careful matching of LoRa configuration settings between them (see top of Appendix H for full settings). A description of the most impactful LoRa parameters is shown below:

1. **Spreading factor:** Determines how much the signal is spread out over the air, higher rates increase range at the cost of a lower data rate and increased transmission time (which affects battery life of transmitter).
2. **Frequency:** Must match across devices - this project uses 868 MHz, the UK LoRa ISM band.
3. **Bandwidth:** Determines how wide the signal is, I am using 125 kHz.
4. **Transmit power:** Affects the power and therefore range. This must be set within legal limits (e.g., 14 dBm in the UK).

9.1.1 Compliance with regulatory limits on radio power

The UK has strict regulations on the usage of radio transmitters under the Ofcom ISM band rules [29]. For the 868 Mhz band, the maximum effective radiated power that can be used is 25mW. This corresponds to a transmit power of roughly 14dBm.

Additionally, the UK has rules on duty cycle rates. This is essentially how long radio signals are permitted to be on air. For example at a spreading factor of 7, a 20 byte message takes approximately 1/15th of a second to send[2]. The duty cycle limit in the UK is 1%, meaning you may only transmit for 1% of the time on a given day. 1% of a day is 864 seconds. This means to stay in line with UK regulations only 864*15 = 12,960 messages can be sent on a given day - or roughly 1 message every 6 seconds. My nodes send a new LoRa packet every minute so are well within this limit.

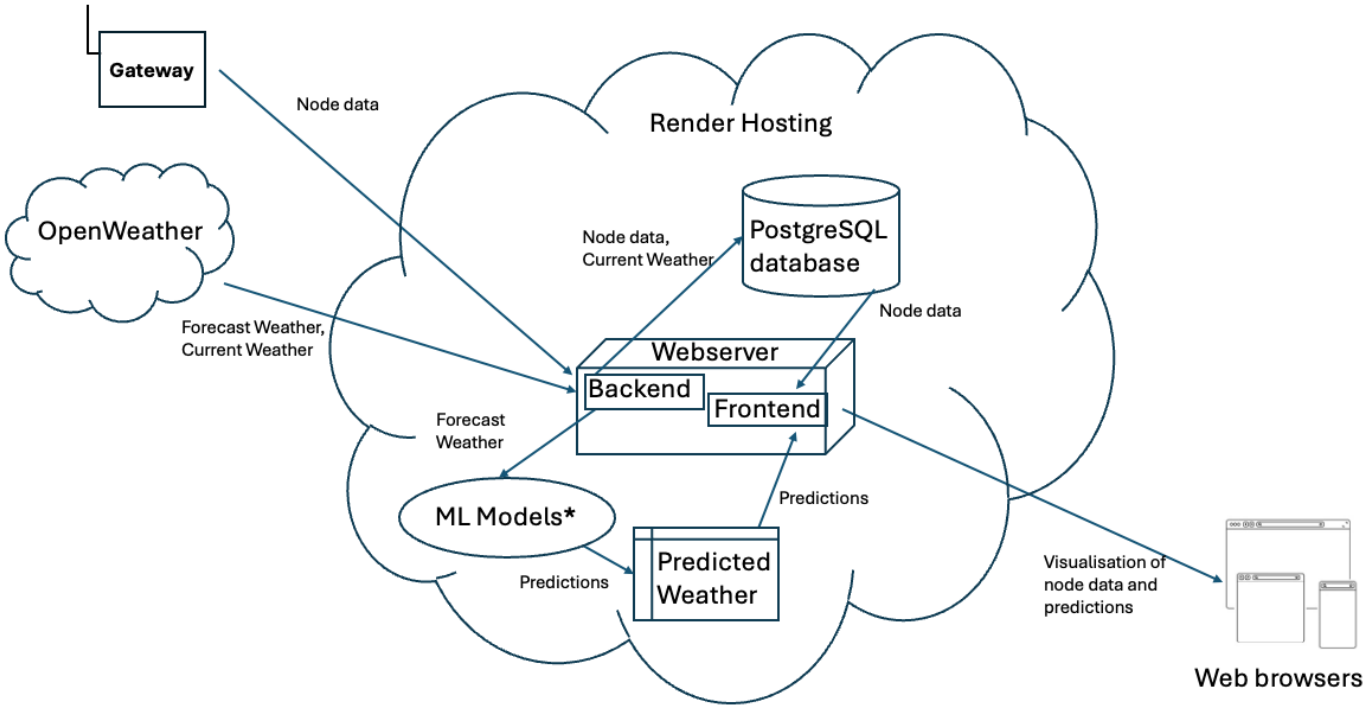
[2]Calculated using this only calculator `https://www.thethingsnetwork.org/airtime-calculator`

9.2 Remote diagnostics and error handling

Since the sensors would not be accessible easily, errors needed to be reported remotely to diagnose faults and problems. Unfortunately, while the sensor nodes and repeater are LoRa enabled, there is no simple way to send software updates to them other than manually. Therefore I developed my code (see Appendix H) with `try/exception` blocks in Python so that if a sensor became disconnected or failed, the exception is caught and the program continues running. Then before the packet is sent out my `find_err()` function scans the payload for errors (which would show as None from the error catching above) and if an error is found then the string `ERR` is sent with the packet. As of the 2nd of September I have so far detected none of these errors in the LoRa packets received.

Diagnostics on the gateway node was luckily much simpler as this was powered 24/7 and network connected. I therefore could remotely access it using a Tailscale virtual private network (VPN) [30]. This created a secure connection, allowing diagnostics from any location without complex firewall configuration.

10 Overview of software design



* Training the machine learning models is not pictured on this graph.

Figure 18: Network diagram of webapp

The overall software design of the webapp is shown in Figure 18. The hosting platform is discussed in Chapter 11. The backend API, database and OpenWeather integration are then discussed in Chapter 12. After this, the frontend is discussed in Chapter 13. Finally, the training and deployment of the machine learning models is discussed in Chapter 14.

11 Hosting

Both the backend and frontend were hosted on Render [31]. I chose Render as a good low cost option for hosting the web service as it offers a free trial period, constant uptime and backups. A feature I particularly liked was the ability for the server to instantly deploy whenever I pushed a commit through to GitHub. If the deployment on Render failed then it would revert back to the previous instance meaning the website was never down.

I have two servers with Render: one handles the database and the other runs the backend and frontend. The database server has 256 MB of RAM; 0.1 share of a CPU and 1 GB of storage. A single instance of a Postgres database is available on this service - which is free for the first 30 days and then costs $5 a month. Meanwhile the backend and frontend instance are run with 512MB of RAM and a 0.5 share of a CPU with no storage as all permanent data is held on the database. This second service costs $7 giving a total cost of about £10. These resources should be sufficient for this project as only small amounts of data are involved. Eventually, I aim to migrate the website to a university hosted server to remove this ongoing cost.

12 Backend

12.1 postgreSQL database

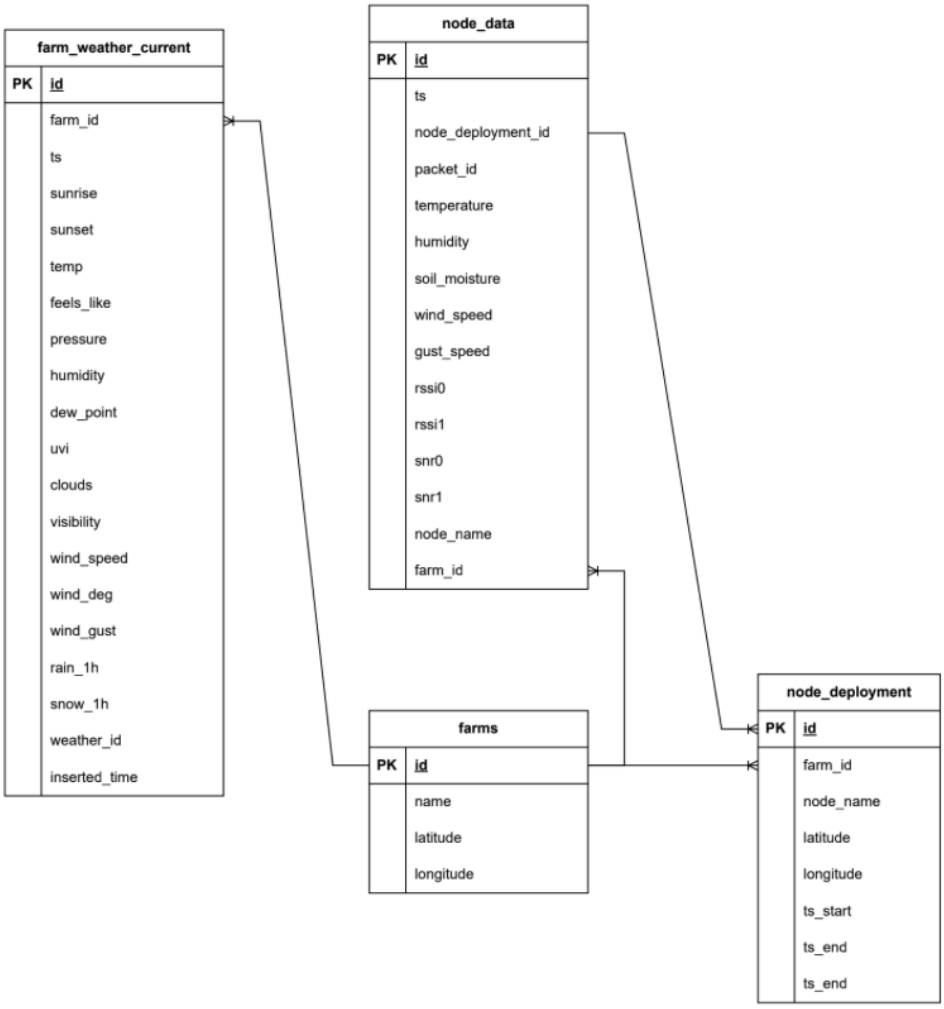


Figure 19: Table schema for postgreSQL database

The database has four tables:

- farms: this holds a list of where the nodes are (or have been) deployed. The longitude and latitude of the farm is then used in API calls to OpenWeather (more on this in Section 12.4) which is used to fetch current and forecasted weather data.
- node_deployment - This holds the precise longitude and latitude of each node as well as information on when the deployment started and ended. If the nodes are moved then this table is updated with new information.
- node_data - Holds the full set of readings from each node.
- farm_weather_current - Holds current weather information from API calls to OpenWeather taken every 10 minutes and is only used to train the machine model explained in the machine learning section.

12.2 Building an API

With the database configured, I then developed a backend API using TypeScript [32], Node.js [33], and Express [34]. This service is responsible for interpreting incoming HTTP requests and performing corresponding database operations - it is essentially the bridge between the gateway hardware and the database. In addition to handling sensor data, the service also integrates with an external weather API in order to retrieve both current conditions and forecast data, which are then stored alongside the locally collected sensor readings as explained in Section 12.1.

The API inserts data into the database when it receives a POST request. The body of the POST request contains a JSON body which is parsed, processed and added to the database by the API. Data can be retrieved for display by the frontend using a GET request.

The entire api is written using TypeScript. TypeScript is JavaScript with added syntax that forces strong typing and features to enable error catching earlier in development. When TypeScript is compiled it produces a JavaScript file which contains the actual code that is then executed. The main benefit of TypeScript is that it leads to much more robust code with vastly fewer typing errors that vanilla JavaScript can often let slip by. As I needed my API to have constant up time and reliably insert and select from my database this made TypeScript ideal for this application.

On top of TypeScript I used Node.js and the Express framework. Node.js allows JavaScript to run on the server instead of inside a client browser, which lets the project share a single language (i.e. TypeScript/JavaScript which are very similar) across frontend and backend development. It is well suited to creating APIs that handle many simultaneous HTTP requests while calling on external APIs. This is because Node runs asynchronously, so when slow database queries are being executed Node will continue listening and processing further requests. This non-blocking behaviour means Node can handle large numbers of concurrent requests at the same time. Additionally the node package manager (called with npm) has a large set of useful libraries and allows my project to be rapidly set up on new computers without having to retrieve required packages from different sources.

Express is a framework specific to Node.js that has tools for creating the API endpoints. I used express to write my request-handling functions for each URL endpoint. These functions process incoming HTTP requests to the endpoint and then execute the SQL query associated with that endpoint. Instead of constantly opening new connections with the database I used the pg.Pool library [35] to pool requests together on the same connection. This helped to improve the speed of SQL lookups and inserts, which accelerates execution and in turn improves the perceived performance of the webapp.

In total my api has five endpoints:

- POST '/api/database/insert-node-data' - This endpoint is used by the gateway to fill the database with sensor data. When a valid HTTP request posts to this endpoint the JSON in the body of the request is parsed and inserted into the node_data table. This endpoint has been included in Appendix I as an example.
- POST '/api/database/insert-weather-data' - This endpoint is used to insert general weather data from the OpenWeather API into the relevant table. The backend calls this every 10 minutes to collect the latest general weather data.
- GET '/api/database/select-latest' - This endpoint sends the latest sensor data from the node_data table to the computer that sent the GET request. It is used by the front end to display current weather data.
- GET '/api/database/select-node-range' - This endpoint sends back the range of sensor data that is specified in the request body. It is used by the front end to build chart data.
- GET '/api/database/forecast' - This endpoint is used by the frontend to retrieve the latest forecast model data from the backend.

12.3 Security

Security was important in the design of the backend API as the end points are exposed to the internet and thus any internet connected device is able to call these. Therefore, I had to take steps to prevent erroneous calls (such as from bots) to my API endpoints which could insert data incorrectly or scrape the contents.

12.3.1 Environment variables

The first step I took was to use environment variables inside Render to hide sensitive information such as API keys and passwords. These variables are hidden from any source documents and are injected separately at deploy time. This prevents the information from leaking into the public domain.

12.3.2 Bearer tokens

Then I needed to prevent any unauthorised machine attempting to perform API calls on my backend. The approach I took to this was to follow guidelines set out in RFC 6750 related to the use of 'Bearer tokens' (essentially a password check) [36]. For this I checked that all incoming HTTP requests to my API had an 'Authorization' line in their headers. I then checked whether the string token after this matched the SECRET_WORD variable in my environment variables. Any request to the database without the correct header and token is automatically rejected.

12.3.3 Protecting against SQL injection

Finally in case those security measures are not sufficient and a malicious actor gets access to my API end points I then added protections against SQL injections. SQL injections are a technique where an attacker can try to manipulate a database by sending a crafted input that alters the intended purpose of the endpoint.

For example if my API end point was designed to allow selections from a database it might naïvely allow queries such as this:

```
SELECT ${httpBody.column} FROM ${httpBody.table_name};
```

Expecting the `httpBody` to contain a JSON with `column = "*"` and `table_name = "node_data"`, for instance.

However, if someone were to name their column as `"*"` and their table_name as `"node_data; DROP TABLE node_data;"` the effective command would change to:

```
SELECT * FROM node_data; DROP TABLE node_data;
```

This command would then select the data as intended before completely destroying the table and all of its data. To prevent this from happening I whitelisted valid inputs like so (psuedo-code for clarity);

```
const white_listed_columns = new Set('*');

const white_listed_tables = new Set('node_data');

if (!white_listed_tables.has(httpBody.table_name)
    or !white_listed_columns.has(httpBody.column)) {
    return 404 error;
}
```

12.4 Weather API integration

The weather API I settled on using to help build data for the forecasting function of the webapp was the One Call API 3.0 by OpenWeather [37]. This API offers current weather data at a 10 minute

resolution as well as 48 hour ahead hourly and 8 day ahead daily weather forecasts. The API permits up to 1000 calls per day before separate payment is required. As I would only be requesting one current weather data reading and one forecast data reading on 10 minute intervals I would only put through 288 requests per day which is well within the free limits.

To set up automatic API retrieval from my backend to the weather API I used the node-cron library [38] to set up a 10 minute job on my host server. Node-cron would call the POST '/api/database/insert-weather-data' endpoint as explained in Section 12.2

13 Frontend webapp

13.1 Design choices

Before creating the frontend, I reviewed a handful of websites that display weather and other data—including the Met Office, BBC Weather, and Apple Weather—and informally discussed the preferred designs with peers.

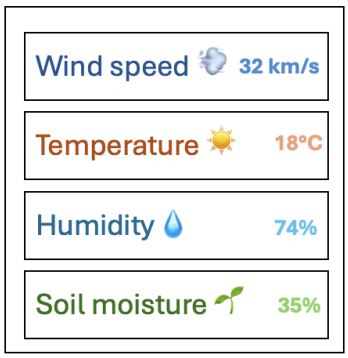


Figure 20: Initial paper prototype for webapp with live information shown in clickable boxes

Ultimately, I settled on a tile-based design similar to the paper prototype in Figure 20, as this offered a simple and intuitive way to view all current information at a glance. The concept was for users to click on any tile to load a detailed chart for that specific data type. The following sections describe the technologies selected to achieve this.

13.2 Choice of language in webapp

The frontend of my project is written in HTML, CSS and JavaScript, which is an archetypal web development stack.

13.3 PicoCSS

PicoCSS is the library I used for the default styling of my webapp. I liked its minimal and low distraction look which I thought was ideal for presenting data. It also had good mobile to desktop scaling which meant my webapp (at least the non-chart parts) worked on mobile and desktop with little work needed.

13.4 Apache Echarts

Apache Echarts is a JavaScript library that I have used to build the charts for my webapp. Data for charts is fetched from the backend and then loaded into a 'series' object. Apache Echarts then

renders the data series onto a graph with a variety of options that I have chosen to improve data clarity.

13.5 Walkthrough of webapp features

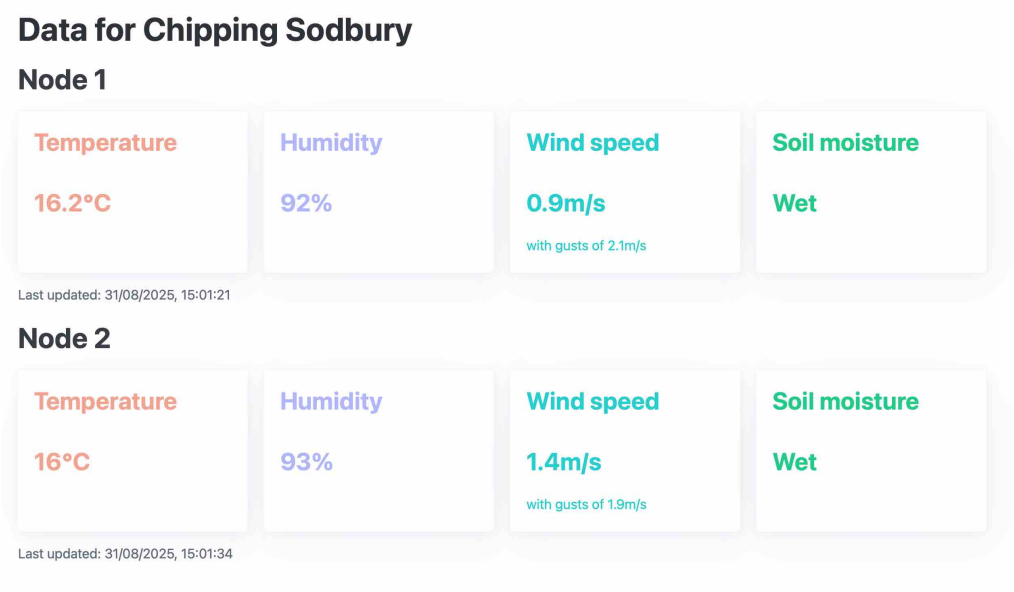


Figure 21: Main page of website showing current sensor readings

Figure 21 shows the main page of the webapp, which displays the most recent readings for each node. A "Last updated" timestamp appears under each set of readings, indicating how recently the data was received. Clicking on any of these tiles opens a detailed chart for that specific metric.

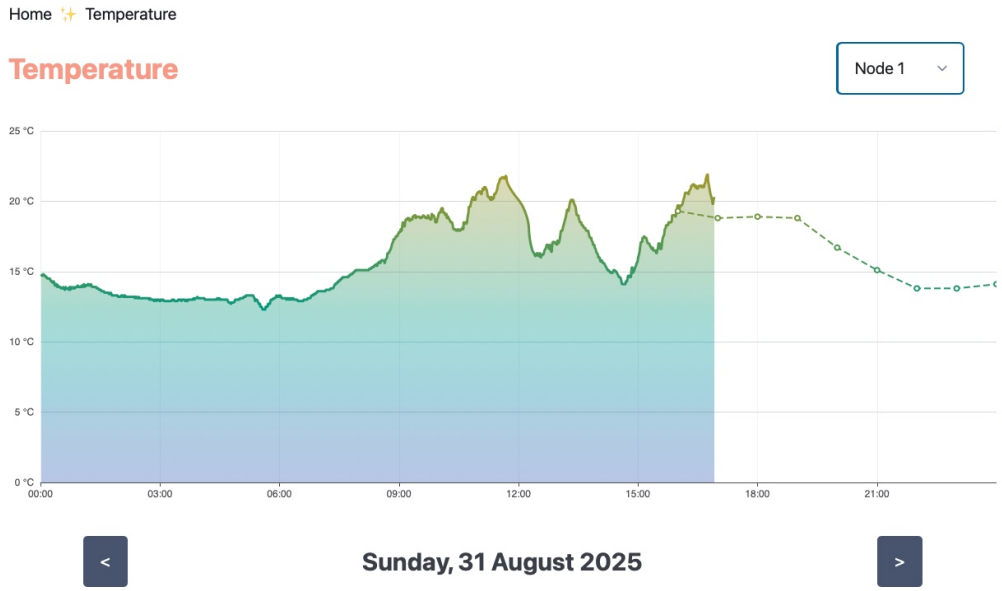


Figure 22: Temperature page with chart showing data for node 1

The temperature page in Figure 22 features a colour gradient across the temperature range, with warmer values appearing more reddish and cooler values more bluish. This is generated by the visualMap() function in Echarts and provides the user with a quick visual cue of the current conditions.

Figure 23: Humidity page with chart showing data for node 2

The Humidity page in Figure 23 similarly has a gradient applied. Another noteworthy page element is the drop down menu in the top right, which allows the user to switch the view between "Node 1", "Node 2", and "Compare". Selecting an option updates the chart to show the corresponding dataset, with "Compare" displaying a combined view of both nodes.

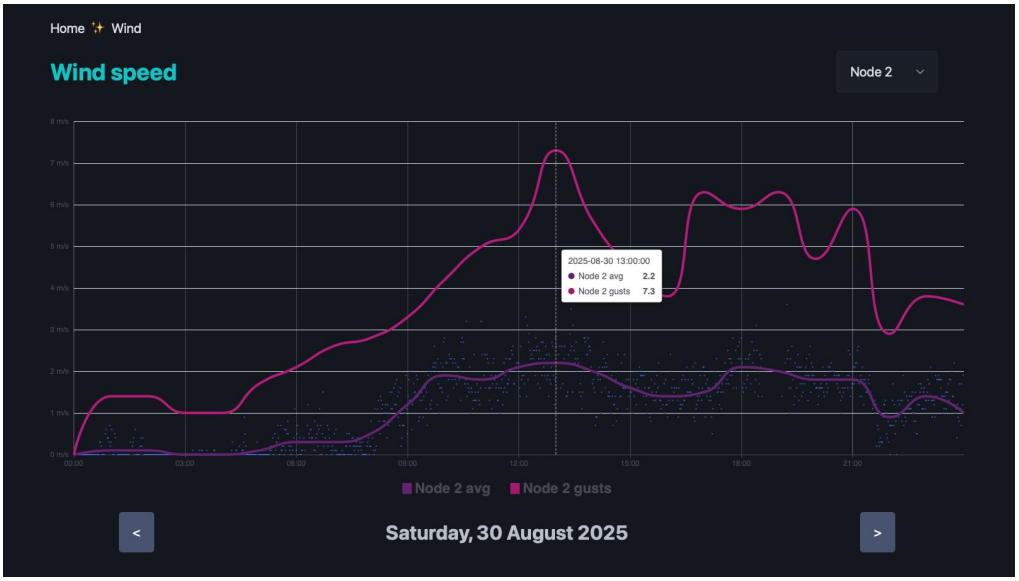


Figure 24: Wind page with chart showing data for node 2

Figure 24 shows the webapp in dark mode. This is a feature of the PicoCSS library, with the site's colour automatically adjusting to the user's preferred appearance settings on their device.

Another key feature shown here is the tooltip. It appears when a desktop user hovers over the chart or if a mobile user taps and drags. This displays the precise numerical values on the chart at the point the user is hovering over.

Compared to the other chart types, the wind speed chart displays three data series: a scatter plot of all raw wind readings (which is difficult to see in the figure), a line for the hourly average wind speed, and another for the highest gust speed recorded each hour. Also below the chart there is a

legend that allows the user to toggle the visibility of the average and gust speed lines.

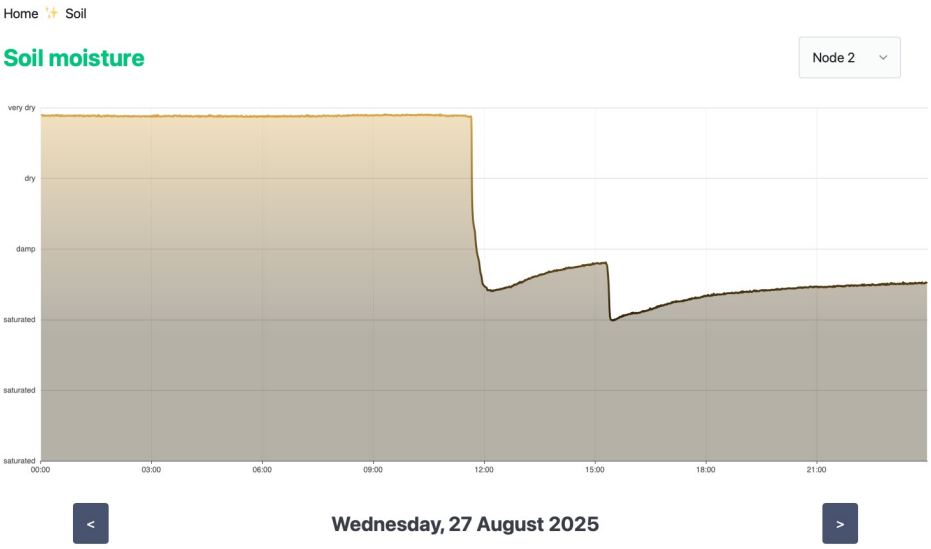


Figure 25: Soil moisture page with chart showing data for node 1

The soil moisture page in Figure 25 shows the first day of heavy rain for the nodes. Other elements not already discussed include the navigation bar in the top left that allows users to click back to home while also showing the current page. There are also buttons to the left and right of the date that allow the user to change date.

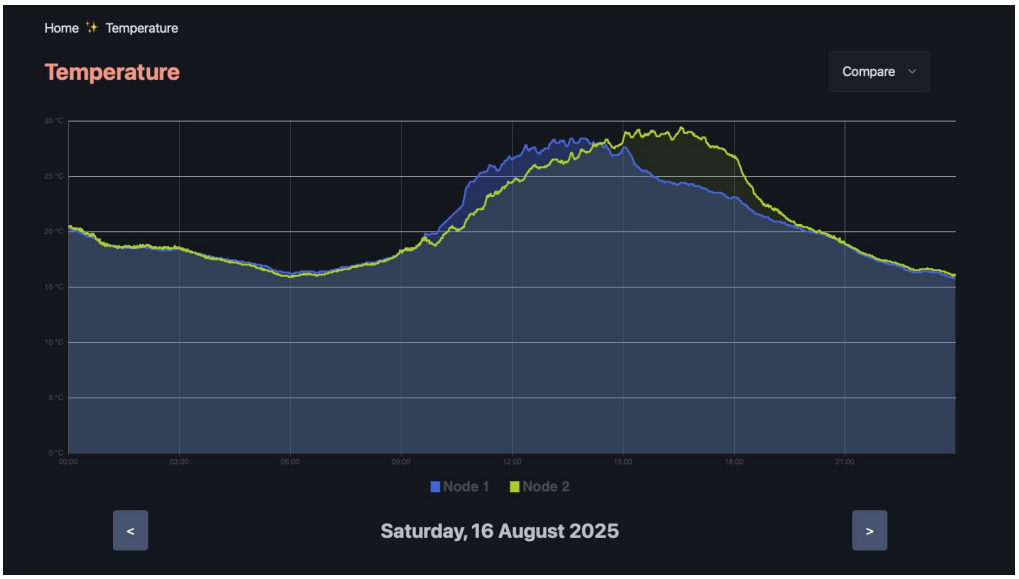


Figure 26: Temperature page showing comparison graph

Figure 26 shows the chart in comparison mode. In this view, each node's data series is assigned a distinct colour to allow for easy visual distinction between them.

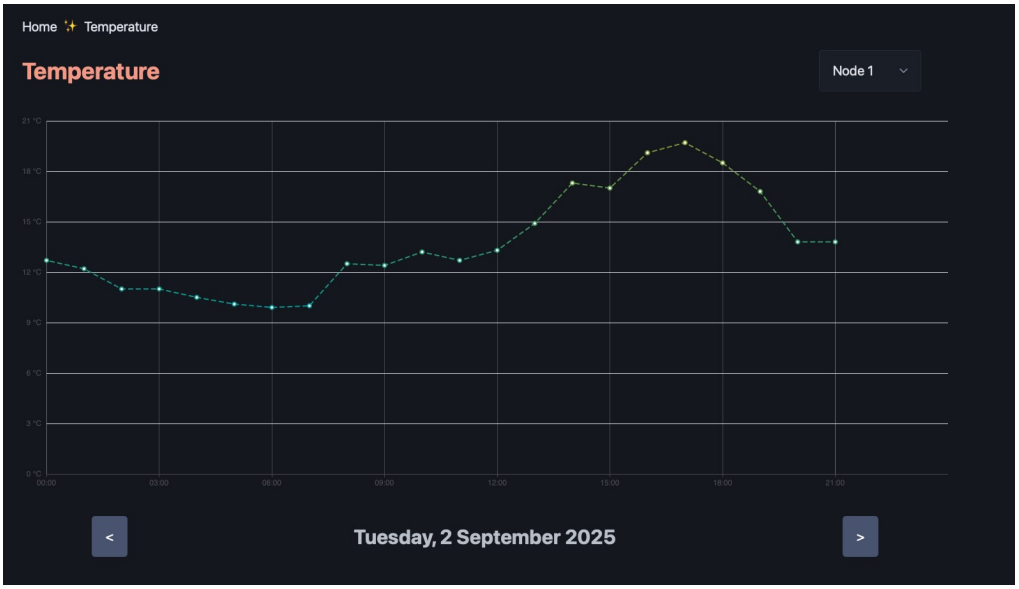


Figure 27: Temperature page for node 1 showing forecasted data

Finally Figure 27 shows the forecasting feature. This applies to future time periods, presenting sensor data forecasted up to 48 hours in advance by a machine learning model.

13.6 Making the app mobile friendly

I wanted the webapp to be accessible for both desktop and mobile users. What tends to make this difficult is the fact that scaling on desktop and mobile is normally very different. Mobile devices have a longer vertical axis while on desktop it tends to be the reverse - so I needed to ensure elements were reactive to the screen size of the viewer.

PicoCSS comes with much of this reactive formatting as standard on default HTML elements (selection boxes, divs, titles, navigation bars etc). However integrating Echarts was difficult as PicoCSS styles cannot directly apply to this. I therefore took actions to improve the usability for mobile users, as summarised below:

1. Dynamic tooltip: The webapp can tell if the viewer is a touch screen and if so it will make the tool tip hover slightly away from the point of touch. While on desktop the user will want to hover directly over a datapoint and see the tool tip appear where they are hovering, on mobile the user's finger would block this information. I fixed this issue by making sure the tooltip hovers slightly away from the selection point on mobile devices.
2. Dynamic chart and font size: Echarts comes with no standard method of resizing chart data depending on the size of the device viewing the chart. So I ensured that the size of the chart (including its font) would be drawn smaller on mobile devices and recalculated if the viewing size changes (i.e. if a desktop user shrinks the window size)
3. Dynamic X axis: The interval between X axis ticks on mobile is 6 hours vs 3 hours on desktop, so that on mobile screens the view is less crowded.
4. Dropdown menu for chart selection: I initially used a radio design for the chart type selection (similar look to multiple choice question buttons) but quickly realised this was not a mobile friendly design choice as the scaled down buttons were very hard to click. I instead opted to use a drop down box as this was much easier to select on a mobile setting while still a well known element that users should be familiar with.

One of the hardest pages to format well for mobile users was the wind sensor comparison page as this has many different series elements to display (two wind averages and two gust averages). However,

from Figure 28 we can see that all elements fit comfortably in frame despite the large number of series and legend data.

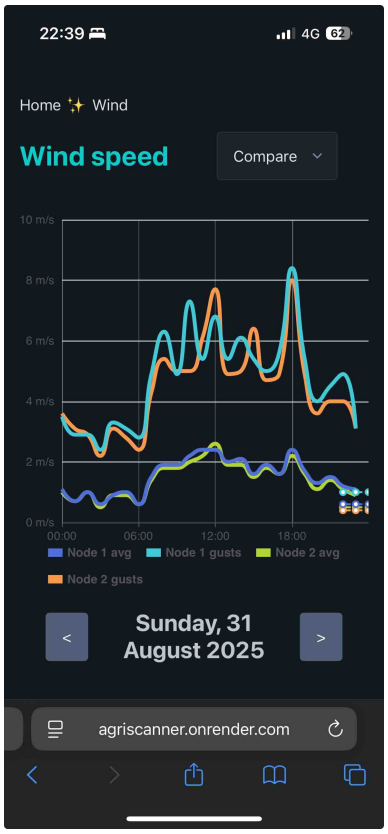


Figure 28: Mobile phone showing wind speed comparison chart

14 Forecasting with machine learning

Once the webapp and database were operational I moved on to developing machine learning models that would attempt to forecast the microclimate datapoints for the following 48 hours based on the general weather forecast data for the region.

14.1 LightGBM

LightGBM is a popular machine learning algorithm developed by Microsoft. I chose LightGBM as it offers a good balance between performance and training efficiency compared to similar models such as XGBoost [39].

Both LightGBM and XGBoost are known as 'Gradient Boosted Machines'. This is a technique in machine learning that effectively combines many weaker machine learning models (called weak learners) to create a single highly accurate model. These models are excellent for tabular data (such as my node and forecast data) and regularly beat out competing learning algorithms [40]. LightGBM is also compatible with the m2cgen library that allows the final model to be converted to JavaScript format, allowing it to work within my webapp. All these factors made LightGBM a good algorithm for my use case.

To train the machine learning models I installed Python with the LightGBM and m2cgen libraries as described above on my personal laptop. I then also installed pandas for data handling purposes.

14.2 Machine learning process

I aimed to create ten separate machine learning models, one for each node (node 1 and node 2) and sensor reading (temperature, humidity, wind speed, gust speed and soil moisture).

Training machine learning models requires inputs (referred to as “features”) and outputs (referred to as “targets”). For my models, the features came from the OpenWeather past **actual** weather data and the targets came from the node sensor readings. While training, the ML model can "see" both features and targets. Figure 29 below illustrates steps in the training process.

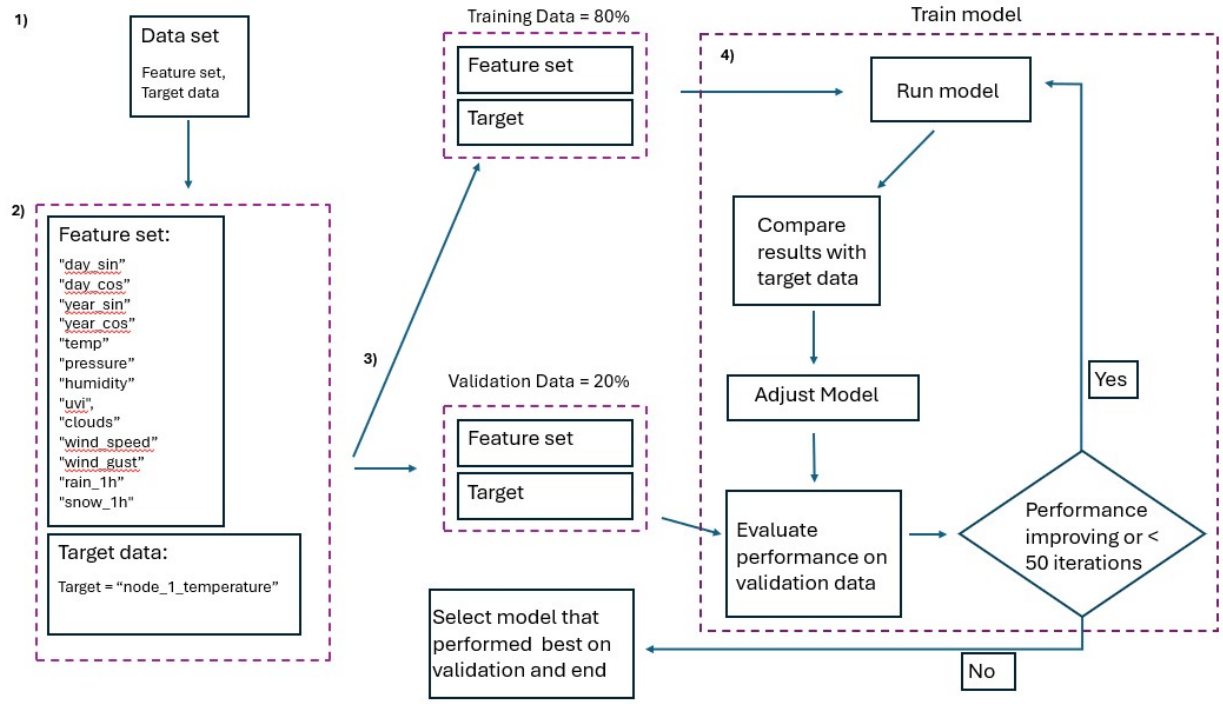


Figure 29: Infographic showing steps for training with LightGBM. This uses the temperature target for illustration, for other sensor readings the target was changed to that reading

The following steps were followed to train each of the ten models.

1. Prepare the dataset: A single cleaned dataset was created by matching timestamps between the api weather data and the sensor node data. As API readings are taken every 10 minutes and node readings every 1 minute, this meant that 9/10 node readings were discarded. The final dataset was roughly 1,400 rows. The data used for training spanned the period 15 - 27 August as that final date was when I trained the model.

2. Define the feature set and target data: The feature set from the weather API and targets from the node data were defined, and unnecessary columns discarded. The database timestamp field was transformed into sine and cosine representations of day and year. This is necessary when training on a time-series data set as the algorithm must be able to understand the cyclical nature of time. For example, using raw timestamps would incorrectly suggest to the algorithm that the times of 23:00 on day 1 and 00:00 on day 2 are 23 hours apart rather than just 1 hour.

3. Split the dataset into training data (80%) and validation data (20%): The data is split by time so the training data consists of the first 80% of the rows and the validation data the last 20%. This data is then supplied to the model.

4. Run the iterative training model: For each iteration, the model looks at the inputs (training features) and the correct answers (training target) of the training rows, and determines where it

is getting incorrect outputs. It builds a small decision tree that specifically aims to correct those mistakes on the training rows and adds that tree into itself so its predictions change a little. It then applies the updated model to the validation inputs (validation features) and compares those predictions to the validation answers (validation target) —to see how well the model would do on new "unseen" data. The validation data are never used to build the tree; they are only used to check the accuracy of the model. If the validation check shows no improvement after a number of iterations, the training stops and the model keeps the version that performed best on validation. The process will perform a minimum of 50 iterations. I set the maximum number of iterations to 250 to prevent the models getting too large, as each iteration increases the model size substantially (The humidity model is over 40,000 lines long in JavaScript format for example).

Once trained, the final models were uploaded to the backend and the inputs then came from the OpenWeather **forecasted** weather data for the next 48 hours as explained in the following section.

14.3 Machine learning deployment

Once the ten models had been trained, they were uploaded to the web server. I wrote an automated function on my backend that provides the models with datapoints from the OpenWeather **forecast** data for the next 48 hours, and updates this data every ten minutes to adjust the predictions as the forecast changes. The outputs from the models are recorded as hourly predictions for each datapoint in a JSON file, which is requested by the frontend software and used to display a line graph of predicted values for the next 48 hours.

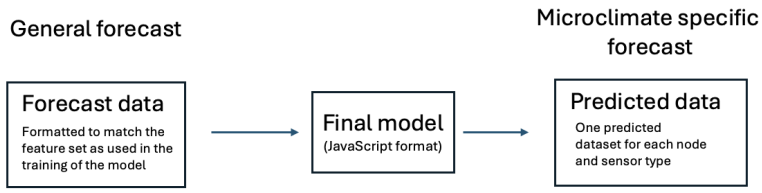


Figure 30: Infographic showing how final model is used on the webapp

Part IV
Evaluation

15 Hardware evaluation

15.1 Qualitative discussion of weather station performance

15.1.1 Battery life, solar power and outages

The nodes have had more downtime than expected based on the energy budgets in Sections 7.2 and 7.3, despite doubling the battery capacity during deployment. The primary cause is site place-

ment, with both nodes in a north-facing garden bounded by 2.5 m fencing and a two-storey house immediately to the south. As a result, they receive no direct sunlight until 09:00, and Node 1 is shaded again by 15:00 due to the shadow of a nearby garage. This leaves only ≈5 h of direct sunlight, which is far below the amount assumed in the design of the nodes.

Outages seem to only affect the sensor nodes; the repeater has had zero downtime since installation. This is consistent with its much lower energy budget from a lack of any peripheral sensors, which reduces average energy draw.

The limitations of the current location, plausibly explain the observed outages. Because these shading conditions are unlikely in an open field, these outages are not of primary concern for the intended deployment. However, it highlights the system's sensitivity to late sunrise and early sunsets from terrain. This would potentially be an issue if the actual deploy site was shaded by a hill or some tall trees late in the day, which I had not properly considered in the design phase.

15.1.2 Weatherproofing

The summer of 2025 is set to be one of the hottest and driest on record with August specifically receiving roughly half its average rainfall [41]. Consequently, from the 15th of August to the 26th of August the weather stations saw only short spells of rain, making an analysis of their weather proofing hard to review up to this point.

Fortunately, from the 27th August onwards the nodes were exposed to fairly heavy sustained rainfall. Throughout this period there has been no evidence of water ingress.

Beyond ingress testing, the nodes experienced elevated ambient temperatures (exceeding 32 °C) and showed no signs of overheating. These results are encouraging, although longer-term deployment that includes colder weather and sustained heavy rain is needed to comment conclusively on their overall robustness.

15.1.3 Range

As reported in Section 7.1 the Challenger microcontrollers and antennas used in this configuration are able to reach at least 1200m of range in a real world situation. The conditions in that test were not ideal for LoRa as both the receiver and transmitter were positioned low to the ground at around 1-1.5m. In addition line of sight was broken at around 1000m which severely diminishes the performance of any radio communication system. There were also buildings between 1200m and 1400m that blocked LoRa signals entirely.

In the intended deployment of my full weather network, the use of an additional repeater would significantly improve range. In a flat terrain setting the use of a repeater would double the range, but if terrain is hilly a repeater can increase range by even larger multiples. The below graphic demonstrates the problem of hills when it comes to LoRa. As a LoRa signal propagates from the transmitter and collides with a hill the majority of the signals energy is reflected off the hill meaning it cannot reach the gateway.

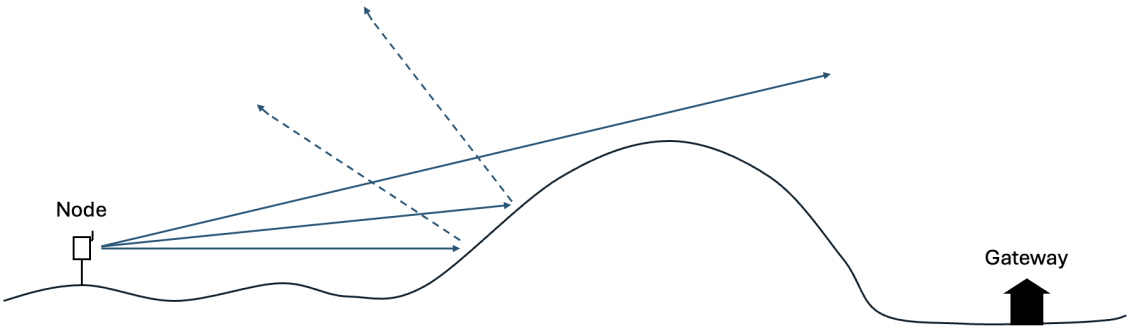


Figure 31: Illustration of LoRa radio propagation between node and receiver with blocking terrain

If a repeater is placed on the top of the hill not only would it mean the LoRa signal would reach the gateway but the repeaters increased elevation would give a significant boost to its range as the total area with a line of sight to the repeater would be vastly increased.

The range that this repeater allows is what sets the design of my network apart from commercial options and is discussed in more detail in the next section..

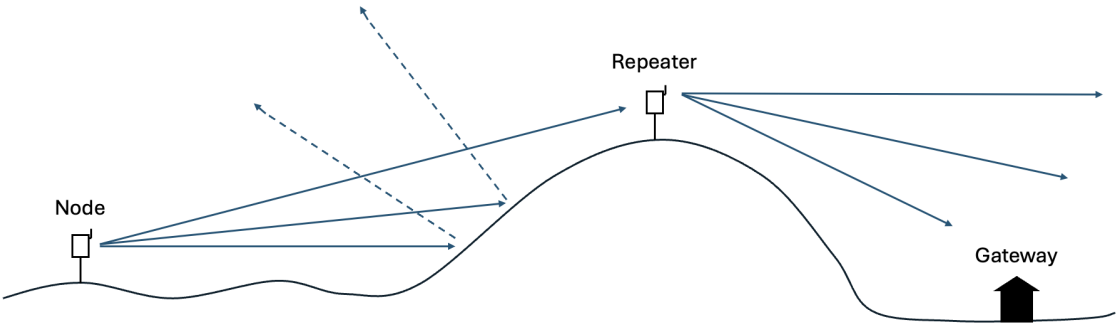


Figure 32: Illustration of LoRa radio propagation with a repeater included

15.1.4 Reset behaviour

As covered briefly in Chapter 8, the reset behaviour of the Challenger is inconsistent after a brownout or complete power outage. Essentially if the sensor nodes run out of power, there is no certainty that the node will restart the code.py program when power returns. This is in spite of the fact that I modified the safemode.py behaviour of CircuitPython with the recommended code for remote solar power applications. As this ultimately appears to be a firmware issue there is not an obvious or easy solution for this.

My proposed solution is to rewrite all of the Challenger software in Micropython and test if that firmware behaves in a more reliable and deterministic way on power outage. The reasoning is that Micropython is a simpler firmware and does not appear to use a "safemode" state. Unfortunately

the issue was picked up fairly late in development and it was unrealistic to perform a refactor of this scale with before submission.

15.2 Quantitative comparison with commercial alternatives

I will now compare the performance of my LoRa weather station to prebuilt options available to purchase. A breakdown of costs for Agriscanner is included in Appendix A.

	Agriscanner Network	**SenseCAP S2120[42]**	**Decentlab Eleven Parameter[43]**	**HOBO weather station kit[44]**	**SparkFun Arduino weather kit[45]**
Number of sensors	4	8	11*	6	7
Sensor accuracy	Hobbyist	Hobbyist	Professional	Professional	Hobbyist
Communication type	LoRa	LoRa	LoRa	Mobile network	WiFi
Update frequency	1 minute	1 hour	10 minutes	1 hour	1 minute
Readings per hour	60	1	6	10	60
Power source included	Yes	Yes	Yes	Yes	No
Power source	Solar	Solar	Solar	Solar	–
Batteries recharge?	Yes	No	No	Yes	–
Reported battery life	replace $\sim$ 3 years	154 days	Several months	replace 3–5 years	–
Reported range	–	2–10 km	2–10 km	Anywhere with 4G	–
Estimated range	2.4–20 km	1.2–10 km	1.2–10 km	–	10–50 m
IP rating	$\sim$ IP65	IPX6	IP66	IP66	None
Ongoing payment?	–	–	–	Yes – mobile plan	–
Ongoing costs p.a	£0	£0	£0	£132	£0
Cost per sensor node	£177	£287	£3,272	£4,138	£130
Cost per repeater	£93	£0	£0	£0	£0
Cost per gateway**	£66	£122	£122	£0	£0
Battery cost p.a.***	£7	£10	£30	£20	£0
Total cost**	**£520**	**£706**	**£6,696**	**£8,418**	**£260**

** Sensors missing from Agriscanner: Solar radiation, rainfall, barometric pressure, vapor pressure, dew point, wind direction, tilt sensor, lightning strike count / distance*

*** For SenseCAP and Decentlab models the lowest cost gateway available is sensecap m2 at £122*

**** Battery cost assumptions are detailed in Appendix C.*

***** Includes sensors, repeater, gateway, and estimated first-year battery + ongoing costs.*

Table 3: Comparison of weather-station options.

15.2.1 Benefits of my weather station

- **Cost:** At £520[3] the Agriscanner weather stations are the lowest cost option among waterproof, long-range systems. While the SparkFun kit is cheaper, it is not suitable for outdoor deployment due to its exposed electronics and, since it relies on WiFi, its range would be insufficient in an agricultural setting. It will therefore not be discussed further. The two professional systems (DecentLab and HOBO) are over ten times the price of my system and are therefore not comparable.

[3]NB: The actual cost for this project is significantly lower as many components are being used on loan from the University of Bristol or purchased through my £150 dissertation allowance. The total charge to the DECIDE project is roughly £200. The larger figure here represents the value of components at market price.

The only system with similar characteristics under £1,000 that I could find is the SenseCAP device. However, with a cost per node roughly 50% higher, my nodes still provide better value. If additional nodes were deployed, this price difference would become even more significant.

- **Frequency of readings:** My sensor nodes collect and transmit readings every minute, providing the Agriscanner webapp with effectively live data. By contrast, most alternative options only provide hourly readings, leaving actual conditions between measurements unknown.
- **Battery recharging:** My nodes use rechargeable 18650 batteries, which I conservatively estimate will last around three years (they use the same battery chemistry as mobile phones) before requiring replacement. Of the alternatives, only the HOBO system features a rechargeable battery. Both SenseCAP and DecentLab rely on disposable alkaline batteries, requiring regular replacement and therefore more frequent maintenance.
- **Range:** As explained in Section 15.1.3, my weather station network benefits from the inclusion of a repeater. This enables significantly greater range than the other models could provide, particularly in hilly terrain.

  I also doubt whether either the SenseCAP or DecentLab systems could achieve a substantial range improvement under the same test conditions. In the UK, LoRa transmission power is subject to a strict regulatory cap (see Section 9.1.1). Since my system already operates at this maximum power, commercial products cannot exceed it. Even with a superior antenna, transmit power would need to be reduced proportionally, as antenna gain also counts towards the effective radiated power. It is therefore highly unlikely that these alternatives would achieve a range advantage sufficient to offset the benefits of a repeater in my system.

15.2.2 Drawbacks of my weather station

- **Number of sensors:** One of the clearest drawbacks of the Agriscanner system is the lower sensor count: only four sensors compared with seven to eleven on competing systems. However, the Challenger microcontrollers in my sensor nodes still have spare GPIO ports, so adding additional sensors would be straightforward. The cost of many sensors is also not prohibitive (for example, a UV index sensor is only £6).
- **Sensor accuracy:** Professional systems use higher-grade sensors with tighter accuracy specs. For example, the DecentLab unit specifies air temperature accuracy of $\pm 0.6\,^{\circ}\mathrm{C}$, whereas the DHT11 used in mine is only rated to about $\pm 2.0\,^{\circ}\mathrm{C}$.
- **Reset behaviour:** As noted in Section 15.1.4, the node firmware can behave unreliably after a full battery drain. While I cannot purchase the other devices in this list to confirm this directly, it is unlikely that the other units would exhibit the same behaviour, especially because their firmware will have been built specifically for remote solar powered applications.
- **Remote Health Diagnostics:** The Agriscanner system currently provides no way to remotely monitor battery health and charging status. Such capabilities would help diagnose issues caused by poor choice of location for effective charging. A location that might have been acceptable at the time of installation might become non-viable as the sun becomes lower in the sky in the autumn and winter, or as vegetation grows and occludes the solar panels.
- **Other factors:** The other devices bring other non-tangible benefits besides their technical specification. For example, warranties, formal testing, vendor support etc. Professional systems commonly offer long-term maintenance and firmware updates beyond the initial purchase, which my prototype solution obviously does not offer.

15.3 Conclusion on hardware performance

While the weather station network has shown promising results as a low cost and long range solution, the unreliable reset behaviour holds it back from being 100% deployment ready. So far the

weatherproofing of the external devices has been encouraging, though testing has been limited due to unusually dry weather and a longer deployment in wetter and colder conditions is needed. The outages seen during the current deployment appear to be an issue specific to the private garden they are in and would be much less likely in an open field environment; however, this does emphasise the need for more extensive site surveys prior to installation. Overall, if the reset behaviour can be addressed and a small number of additional sensors are added, the complete system would be well suited for long term unattended deployment on a farm.

16 Web-app evaluation

This Chapter will evaluate the usability of the Agriscanner webapp using results from a System Usability Scale (SUS) survey that was carried out via online form in late August.

16.1 Procedure

Before answering the main section of the survey, participants were asked to consent to some privacy wording (see Appendix D). Additionally, to avoid stricter GDPR handling I did not record identifiable information (e.g. name, email, age), which I also hoped would encourage more honest responses. I also made sure to ask what device they were viewing the project on (either mobile or desktop) as I wanted to test whether there was any difference in usability between the two.

Participants were then asked to perform a set of four representative tasks in the webapp. Each task involved collecting a piece of information from a specified chart, such as finding the temperature for a particular node at a a specific time (tasks list shown in Appendix D). Once the tasks were completed, I asked participants to submit their answers for each task activity via multiple-choice questions.

Next, the survey presented a standard 10-question SUS, which is a Likert-scale questionnaire commonly used to report on the usability of software systems [46]. The SUS gives a score from 0 to 100, where a higher score indicates that a system is more usable. A "good" SUS score is generally regarded as anything above 68 [47].

Finally, there was an optional textbox for participants to fill out asking for feedback on bugs or features they would like to see. I included this to generate user stories for future development (Section 16.5).

By asking participants to perform the same tasks and collecting their answers, I ensured everyone completed a valid interaction with the webapp before rating it. This helped to standardise the experiment and reduce uncontrolled variance that a more open “try out my website and answer this survey” approach would have. Because I recorded the task responses, I could also compare them to the correct answers, providing an additional metric (task success) to complement the SUS.

In total, 15 participants completed the survey. Participants were recruited mainly from friends, family and other students on my course.

16.2 Hypotheses

16.2.1 Hypothesis 1

The sample SUS score will be greater than the benchmark value of 68

Statistic to test: One-sample, right-tailed Wilcoxon signed-rank test[4].

Null and alternative hypotheses:

$$H_0 : \ m = 68 \qquad \text{vs} \qquad H_a : \ m > 68,$$

[4]For rationale and sources on statistical testing refer to Figure 44 in the Appendix

where m is the population median.

16.2.2 Hypothesis 2

The SUS scores for users on mobile will not differ significantly from those on desktop.

Statistic to test: Two-sample, two-tailed Mann-Whitney U Test

Null and alternative hypotheses:

H_0 : the two populations are equal $\quad$ H_a : the two populations are not equal.

16.3 Results

This section summarises the key results from the survey. Refer to Table 8 in the Appendix for data per participant.

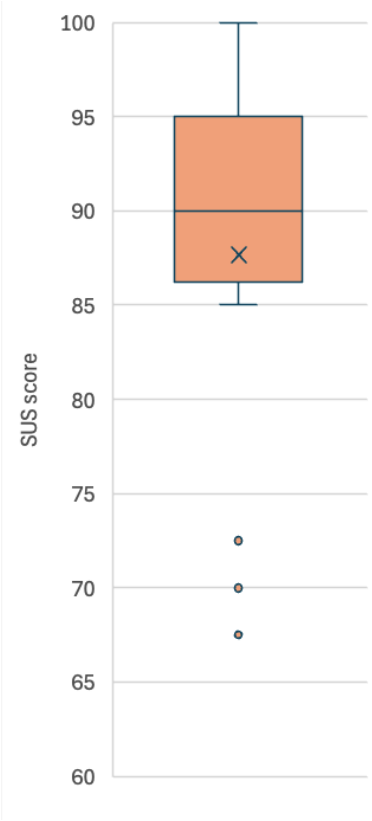


Figure 33: Box and whisker plot showing range of SUS scores from participants

Metric	Value
Maximum	100.0
Quartile 3	95.0
Median	90.0
Mean	87.7
Quartile 1	86.3
Minimum	67.5

Table 4: Summary statistics for SUS

Group	N	Mean SUS
Desktop	7	84.3
Mobile	8	90.6

Table 5: Mean SUS score by device

Figure 33 shows a tight distribution of values between the first and third quartiles, with most participants giving favourable ratings. There were also a smaller number of lower ratings shown as outliers, with a minimum score 67.5.

Table 4 shows the key metrics from the survey. The mean score of 87.7 is well above the benchmark of 68 and indicates a high perception of usability for the Agriscanner webapp. A one-sample Wilcoxon signed-rank test confirmed that the median SUS score was significantly greater than 68 ($p = 0.0004$), allowing for the rejection of the first null hypothesis H_0.

Table 5 indicates that participants using mobile devices gave slightly higher usability ratings (90.6) than those on desktop (84.3). However, a two-sample Mann-Whitney U test revealed no statistically significant difference between the two groups. ($p = 0.105$). Therefore, the second null hypothesis H_0 could not be rejected at the 5% significance level.

Question no.	Incorrect answers	Percentage incorrect
1	3	20%
2	1	7%
3	1	7%
4	0	0%
Overall percentage incorrect		8%

Table 6: Number of incorrect answers for task quiz

Performance in the assigned tasks was generally good (Table 6), with participants answering incorrectly only 8% of the time. Questions 2, 3 and 4 were answered correctly by almost all participants, with only one participant failing on 2 and 3. The first question proved the most difficult to perform accurately with 3 candidates failing. I decided to plot the SUS score given against the percentage of correct answers (see E), however with an R^2 value of 0.0066 there was effectively zero correlation between the two; even if there was some correlation the small sample size and ceiling effect from so many respondents getting perfect scores would make this difficult to detect.

16.4 Discussion and limitations of results

With a very high mean (87.7) and median (90) SUS score, the results here suggest a high degree of usability. A one-sample Wilcoxon signed-rank test helps confirm that the median score of my sample is significantly greater than the benchmark of 68 and cannot be attributed to randomness with $p < 0.001$. This provides quantitative evidence that the frontend development choices I have taken (Chapter 13) have given my webapp excellent perceived usability. This point is further reinforced by the very high task completion rate (92%). The fact that users could not only perceive the system as easy to use but could also operate it efficiently with minimal guidance validates the chosen design approach — particularly the emphasis on a minimal tile based interface that follows user experience norms.

Additionally, the data suggest that usability on mobile devices is no different to that on desktop devices. A two-sample Mann-Whitney U test revealed no statistically significant difference in usability ratings between mobile and desktop users with $p > 0.05$. This is evidence that the steps I took to improve mobile user experience had the desired effect.

Not all the data were positive however: Two participants gave scores close to the benchmark level of 68 and one respondent gave a score below this point. With such a small sample it is important not to dismiss these as outliers. Fortunately, many participants left detailed feedback on usability issues they encountered (Figure 43)- including all three who gave lower scores (Figure 42). These comments provided useful insight into specific usability issues, such as the difficulty with date selection which directly explains the 20% failure rate on the first task. This qualitative feedback was then used to generate the feature requests compiled in Table 7.

There are a number of important limitations to these findings. The sample was small and non-random so the results are not necessarily representative of the wider population. Because I knew many participants personally, social-desirability bias was likely to have inflated ratings. Likewise, a

high proportion of computer-science students in the sample means these participants may have been more familiar with web interfaces than the general public, which could also have contributed to the higher SUS scores. While I measured the completion rate for the user tasks I did not measure the time taken to do this. Even if users successfully completed the tasks it may have taken a long time to do so, which would have been a valuable metric to include here.

In conclusion, while the limitations of the sample mean the results cannot be broadly generalised, they still provide a strong, positive indication of the webapp's usability. An exceptionally high SUS score and a high task completion rate demonstrate that the core design of the application succeeded in its aims. Further, the evaluation has successfully identified specific feedback from users, which provides a clear path to improving usability in future development cycles.

16.5 User stories from SUS survey

Based on the qualitative feedback, several key themes emerged. The most frequently mentioned issues have been summarised in Table 7 and translated into user stories to guide future development.

Category	Mentions	User story
Calendar with date-range selection	4	As a user I want the ability to select date ranges so that I can quickly jump to past dates without repeatedly clicking through days.
Clearer way to find compare graph	2	As a user I want the compare graph to be more explicitly signposted so that I can easily find it when trying to compare nodes
Human readable soil moisture readings*	1	As a user I want an option to view soil moisture in a human readable format (e.g. wet/dry) so that I can understand what the sensor values mean.
Tooltip granularity on chart	1	As a user I want smoother control when using the tooltip without snapping to a particular minute so that I can select more precise times reliably.
Way to export data	1	As a user I want a way to an export weather data to a CSV file so that I can have offline access to it.
Switch between measurements without going back	1	As a user I want a control from within each sensor section (temperature, humidity etc) that allows me to switch between measurement types quickly so that I don't have return back to the home page between each navigation.
About page	1	As a user I want an "About" page describing the project and the data sources so that I understand the context of the data.
Specific issue with chart	1	As a user I want the Y axis to align with my screen properly so that no text is cut off and I can read the labels

*This has now been implemented on the webapp

Table 7: Compiled feature requests from survey

17 Machine learning model evaluation

This chapter evaluates the predictive performance of my machine learning model. The model's forecasts are benchmarked against both actual sensor data and the predictions from a simpler alternative model, before discussing the results.

17.1 Models being compared

The three models that I compared against the actual sensor readings are included below.

1. General forecast weather from OpenWeather
2. Proposed machine learning model (Chapter 14)
3. Alternative model - an adjusted version of the OpenWeather using a mean average adjustment

17.2 Procedure

To evaluate the models I compared the results from all three models over the same 48 hour period (which is the limit of the OpenWeather forecast). The measurements started at 00:00 on Saturday, 30 August 2025, and concluded at 23:00 on Sunday, 31 August 2025.

The OpenWeather forecast and machine learning prediction were collected automatically by my backend (using node_cron and endpoints, see Section 12.2) and stored in a temporary database table I made specifically for this evaluation.

The alternative model was made by comparing the average temperature, humidity and wind speed differences between my raw sensor data and OpenWeather current weather readings from my database[5]. An average difference was then calculated between the two datasets and applied to the raw OpenWeather forecast. The aim of this was to adjust the forecast readings to be closer to the sensor readings. To make the comparison fair with the machine learning model, I used data from 15–27 August (the same as the ML's training data) so that the alternative model would not have a larger amount of data. The purpose of including the alternative model was to assess whether a simple bias correction model was any different to my more complex machine learning procedure.

17.3 Charts and results

As the weather data for the selected period was very similar between both nodes, I have decided to only show charts for node 1 as it is representative of node 2 as well. I have included a table containing the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) for the data in these charts in Appendix K.

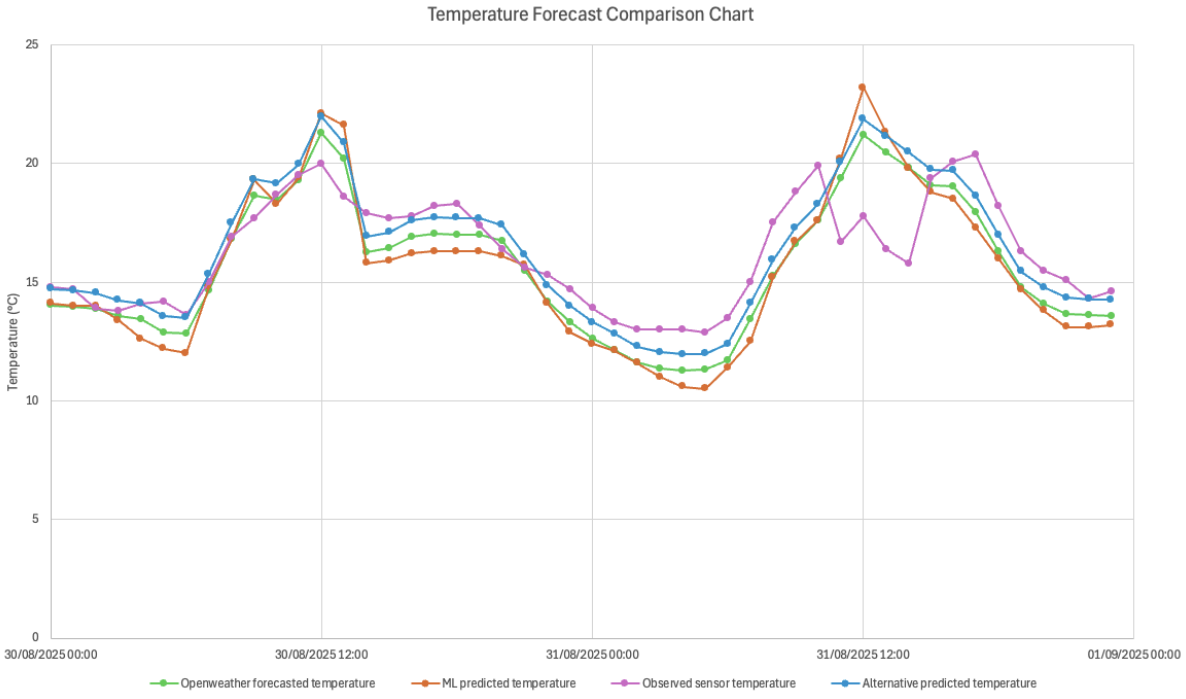


Figure 34: Chart comparing temperature forecast models

[5]See Figure 50 in Appendix for adjustment factors

While Figure 34 does show that the machine learning model (orange) roughly tracked the observed readings, it does not appear to be substantially more accurate than either the general forecast (green) or the alternative model (blue). The relative MAE (see Appendix K) of the machine learning model here is 2.9% while the general forecast is only slightly higher at 3.4%[6], meaning the ML model is only marginally better than its input data at predicting temperature. The alternative model performs better with an MAE of 1.3%. Both the ML and alternative models predicted a higher peak temperature on both days than either the sensor data or the forecast, suggesting that the data from 15th - 27th August may not have been a representative sample and is biased towards higher temperatures.

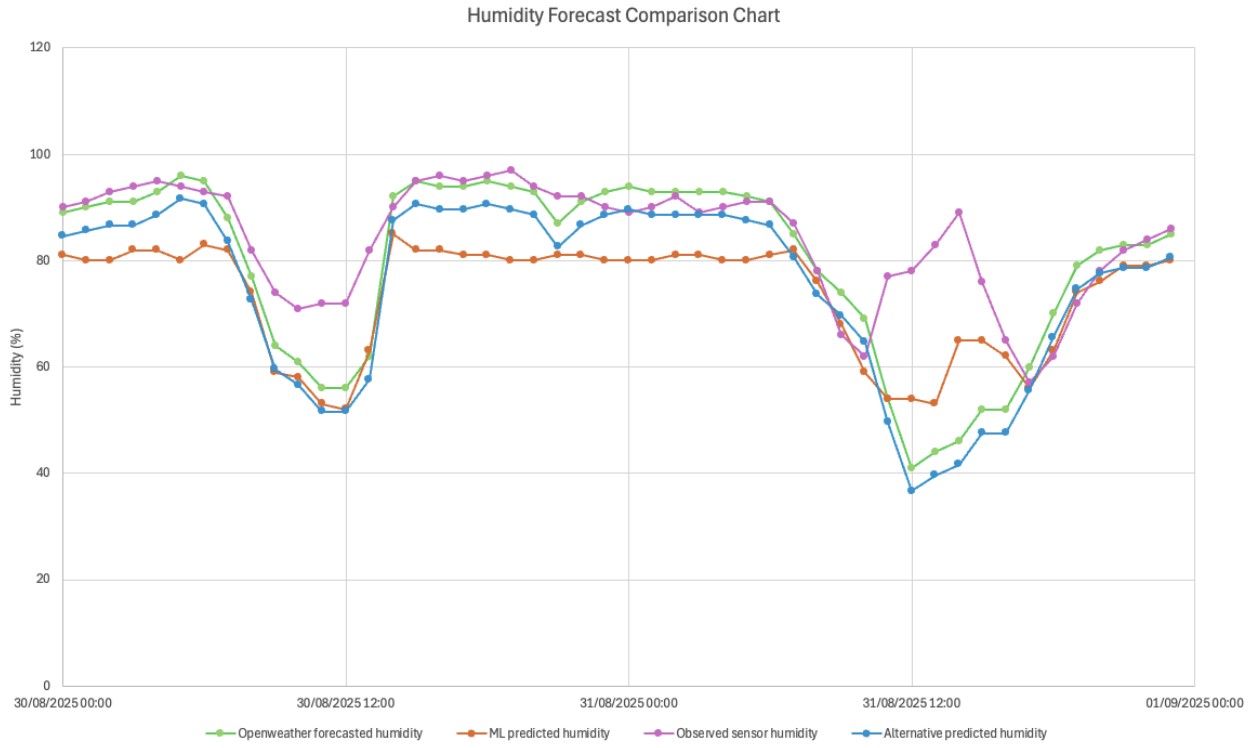


Figure 35: Chart comparing humidity forecast models

In Figure 35 the machine learning model performed poorly on average with an MAE of 14.9% (General forecast: 5.5%). The alternative model also performs worse than the general forecast with MAE of 11.1%. While these metrics paint a poor picture of the model, I would point to the section of the chart around 12:00 31/08. Here we can see that the machine learning model is the only one of the models to successfully predict a spike in humidity at midday. While the magnitude of this spike is not correct it does offer some tentative evidence that the model can more accurately predict some data patterns than general macroclimate forecasts.

[6]MAE scores should be interpreted as follows: a higher score means that the error compared to the actual sensor reading was greater. A lower score indicates a closer (better) relationship

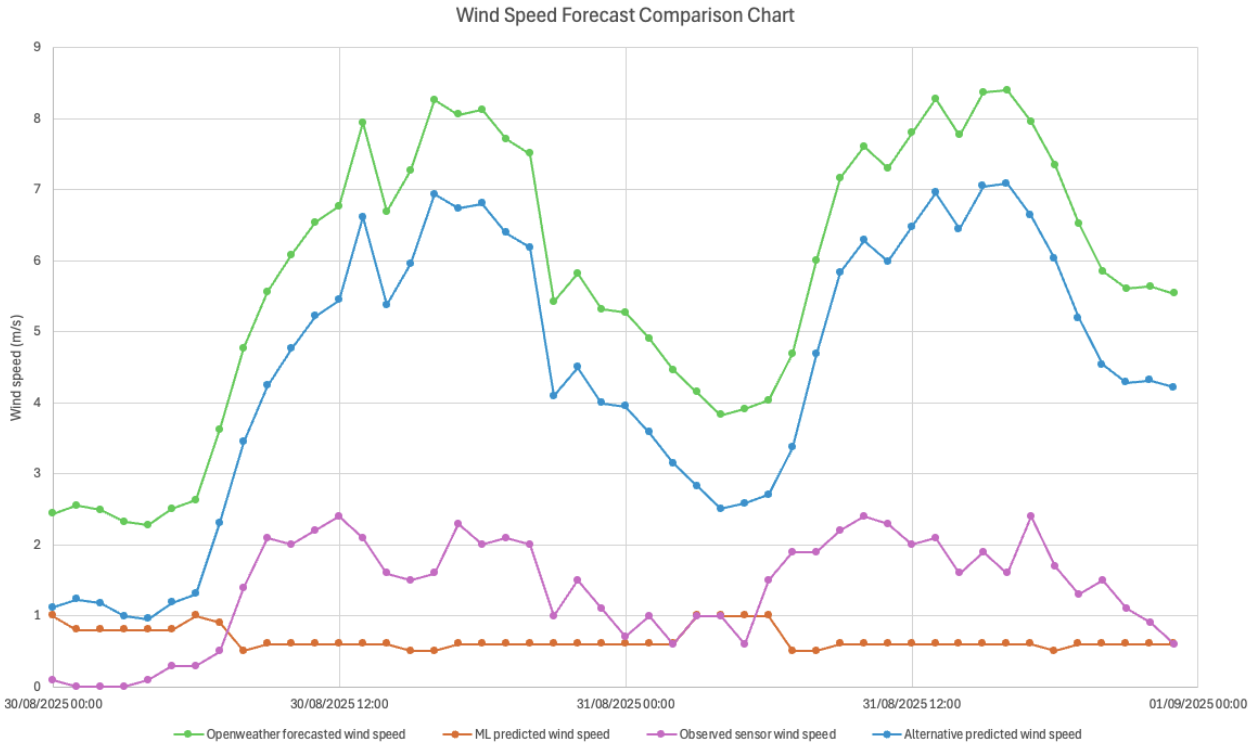


Figure 36: Chart comparing wind speed forecast models

With wind speed (Figure 36) the ML model performs better relative to either the general forecast or the adjusted forecast used in the alternative model. This is reflected in an MAE of 43.6% for the machine learning model versus 381.9% and 264% for the general forecast and alternative model respectively. With that said, the machine learning model does not show any "hump" at midday meaning that the model is not predicting the pattern of wind speed well. This suggests the training data had fewer windy days than the 48 hours evaluated here

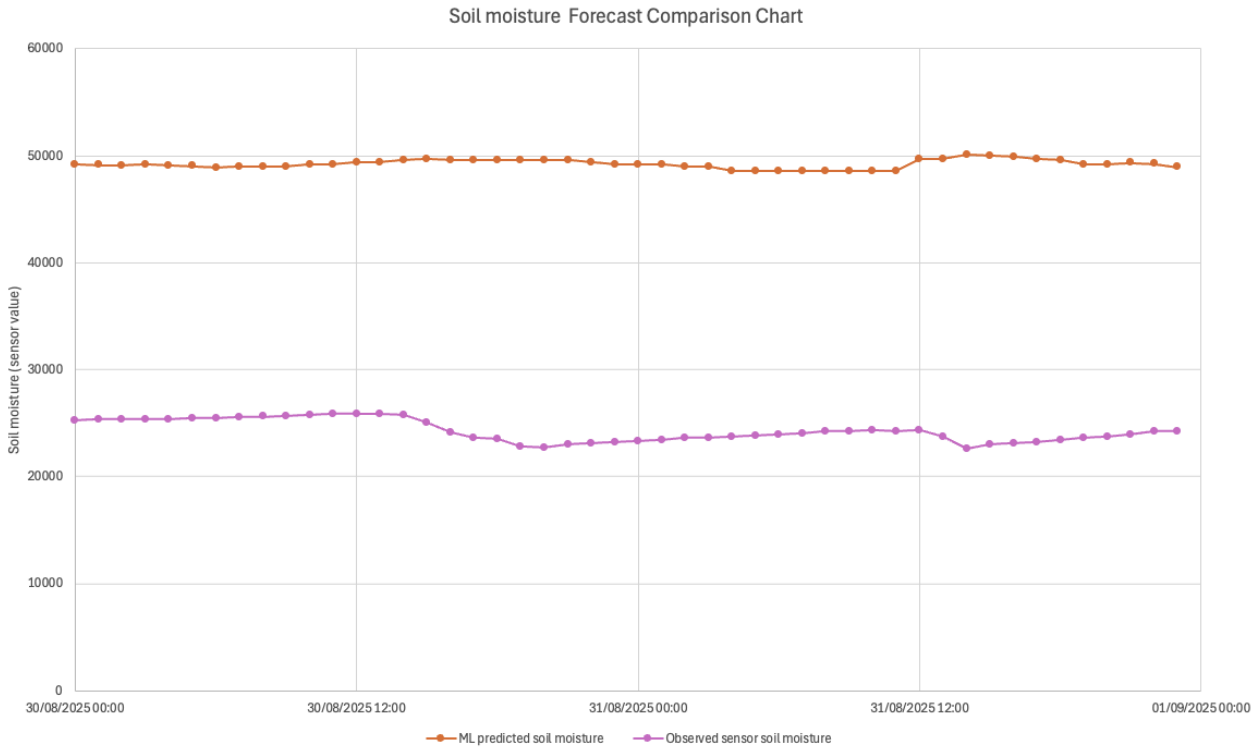


Figure 37: Chart comparing soil moisture predicted by machine learning model versus actual readings

The soil moisture chart (Figure 36) only has the data for the machine learning model and the observed data because the general forecast and alternative models do not capture this metric. The large MAE of 162% here is not surprising at all however - the model's training data was all from an extremely dry period of time and therefore the model does not adjust soil moisture for recent precipitation. The days before the 48 hour period used in these charts had included a lot of rain so the soil was still still wet from this. However the model still expected dry soil as no connection between rainfall and damp soil had been established in the training.

17.4 Discussion of results

While the results for the machine learning model are mixed, there are some bright spots. For example, the model still broadly tracks the actual sensor readings for both temperature and humidity. With respect to temperature, while the MAE of 2.9% was not the best score, in Celsius this represents an error of less than 0.5°C. If we only look at the temperature error for node 2, the error was even smaller at around 0.2°C (node 1 was 0.7°C), and it was the best performer of the three in this context.

Additionally, the humidity chart shows that the model can detect some patterns that the general forecast does not, as seen in the brief reversal in humidity on day 2. Wind speed and soil moisture were clear wins for the model, but mostly due to the highly inflated figures for wind in the forecast and the lack of any soil moisture forecast data.

Despite these advantages, on average the alternative model marginally outperformed the machine learning model in temperature, and both models were surprisingly worse than the general forecast in the humidity context. This raises the question of whether machine learning is "worth it" if a computationally simpler model performed at around the same level.

I would argue that yes, using machine learning is worth it, for two reasons: The first is around the training data: the machine learning model here was not trained on nearly enough data to provide a fair evaluation of its microforecasting abilities compared to the other models. This is most obvious from the soil moisture readings, where the model had no prior rain data meaning it would have been

impossible to predict that soil moisture levels would be lower after rainfall. The general weather conditions from the 15th to the 27th were not particularly representative of the conditions after this point, as the weather cooled significantly past this point with higher average cloud cover and humidity.

The second point is that the machine learning approach is capable and likely to improve, unlike the other solutions which are fundamentally not predicting the microclimate and increased data will likely not improve their accuracy. For example, the alternative model is essentially a blunt correction to the general forecast: even as the dataset grows this approach would become increasingly poor. If in summer the microclimate tends to be hotter and in winter it tends to be cooler then using a simple mean average correction would not only result in no change to the forecast and therefore no bias correction at all.

Hence I believe that with a longer deployment and more data it is highly likely that the machine learning approach would show much greater accuracy than the other approaches, and is one of the interesting avenues for future work (Section 18).

18 Future work

Ultimately the main objective of this work is to deploy the system in an apple farm to assist farmers with the management of their orchard. Before the system can be deployed, the Challenger's firmware must be able to reliably restart after a power outage. As suggested in section 15.1.4 I would first recommend experimenting with MicroPython to see if this can provide a more robust solution to coping with this scenario. The worst case scenario would be to have to use an alternative embedded processor that can handle power fluctuations.

Additionally, down the line it may be of value to add more sensor to the nodes as this is a noticeable weakness of the current system architecture compared to existing alternatives. Considering the addition of sensors would result in increased power requirements I believe it would be inadvisable to add any more for time being. Once the system has been deployed for an extended period and shows that it can operate with minimal outages then the adding sensors would be a worthwhile improvement to the current design.

Once the reset behaviour issue has been resolved, the next stage will be to deploy the system onto a farm. The siting of the sensor nodes and repeater will need careful consideration, so a thorough site survey should be undertaken prior to this. It would be advisable to allow a few days for the installation, so there is someone on hand to deal with any issues that might arise once the nodes have been installed and to make sure the farmers are comfortable with using the webapp.

I would recommend that the forecasting feature continues to be developed using machine learning. Furthermore instead of training the model offline and manually inserting models into the webapp, future work could focus on moving model training online. This would allow the model to continually update and improve as new data from the nodes is produced. After a few months of data is collected then the evaluation in Chapter 17 should be performed again to analyse if the prediction quality has improved.

19 Closing remarks

This dissertation has shown the process of developing a complete IoT weather station network with an accompanying webapp. In the hardware evaluation it was shown that the system designed here offers superior range at lower cost compared to commercial options. The system has continued operating despite rainfall and high temperatures, suggesting the design here is relatively well weather proofed.

The webapp designed here scored well in terms of usability, achieving a far higher SUS score than the benchmark level of 68. The user stories developed from survey responses also offer a clear roadmap to future development.

Finally the machine learning evaluation explored the accuracy of the current machine learning system against a simpler model. With additional time in the field and training data, it is expected that the models accuracy will see large improvements.

References

[1] A. Spiess, *296 LoRa Propagation, Range, Antennas, and Link Budget (incl. LoRaWAN)*, Accessed: 2025-08-18, Nov. 2019. [Online]. Available: https://www.youtube.com/watch?v=BOc3N3Yl38o.

[2] R. Lie, *Lora*, Accessed: 2025-08-18. [Online]. Available: https://lora.readthedocs.io/en/latest/.

[3] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2010.05.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128610001568.

[4] M. Burhan, R. A. Rehman, B. Khan, and B.-S. Kim, "Iot elements, layered architectures and security issues: A comprehensive survey," *Sensors*, vol. 18, no. 9, p. 2796, 2018. DOI: 10.3390/s18092796.

[5] M. S. Farooq, S. Riaz, A. Abid, K. Abid, and M. A. Naeem, "A survey on the role of iot in agriculture for the implementation of smart farming," *IEEE Access*, vol. 7, pp. 156 237–156 271, 2019. DOI: 10.1109/ACCESS.2019.2949703.

[6] T. N. Gia, L. Qingqing, J. P. Queralta, Z. Zou, H. Tenhunen, and T. Westerlund, "Edge ai in smart farming iot: Cnns at the edge and fog computing with lora," in *2019 IEEE AFRICON*, IEEE, 2019, pp. 1–6.

[7] R. K. Kodali, S. Yerroju, and S. Sahu, "Smart farm monitoring using lora enabled iot," in *2018 second international conference on green computing and internet of things (ICGCIoT)*, IEEE, 2018, pp. 391–394.

[8] Met Office, "Factsheet 14: Microclimates," Met Office, Tech. Rep., 2023, Accessed: 16 June 2025. [Online]. Available: https://www.metoffice.gov.uk/binaries/content/assets/metofficegovuk/pdf/research/library-and-archive/library/publications/factsheets/factsheet_14-microclimates_2023.pdf.

[9] World Meteorological Organization (WMO), *Guide to Instruments and Methods of Observation: Volume III – Observing Systems*, 2024 edition. Geneva: World Meteorological Organization (WMO), 2025, ISBN: 978-92-63-10008-5. DOI: 10.59327/WMO/CIMO/3. [Online]. Available: https://library.wmo.int/idurl/4/68661.

[10] S. Yang, L. L. Wang, T. Stathopoulos, and A. M. Marey, "Urban microclimate and its impact on built environment–a review," *Building and Environment*, vol. 238, p. 110 334, 2023.

[11] D. Lai, W. Liu, T. Gan, K. Liu, and Q. Chen, "A review of mitigating strategies to improve the thermal environment and thermal comfort in urban outdoor spaces," *Science of The Total Environment*, vol. 661, pp. 337–353, 2019, ISSN: 0048-9697. DOI: https://doi.org/10.1016/j.scitotenv.2019.01.062. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0048969719300683.

[12] H. A. Cleugh, "Effects of windbreaks on airflow, microclimates and crop yields," *Agroforestry Systems*, vol. 41, no. 1, pp. 55–84, 1998, ISSN: 1572-9680. DOI: 10.1023/A:1006019805109. [Online]. Available: https://doi.org/10.1023/A:1006019805109.

[13] N. Haider, C. Kirkeby, B. Kristensen, L. J. Kjær, J. H. Sørensen, and R. Bødker, "Microclimatic temperatures increase the potential for vector-borne disease transmission in the scandinavian climate," *Scientific Reports*, vol. 7, 2017. DOI: 10.1038/s41598-017-08514-9.

[14] B. Drepper, B. Bamps, A. Gobin, and J. Van Orshoven, "Strategies for managing spring frost risks in orchards: Effectiveness and conditionality—a systematic review," *Environmental Evidence*, vol. 11, no. 1, p. 29, Sep. 2022, ISSN: 2047-2382. DOI: 10.1186/s13750-022-00281-z. [Online]. Available: https://doi.org/10.1186/s13750-022-00281-z.

[15] L. P. Blunn et al., "Machine learning bias correction and downscaling of urban heatwave temperature predictions from kilometre to hectometre scale," *Meteorological Applications*, vol. 31, no. 3, e2200, 2024.

[16] P. Kumar, R. Chandra, C. Bansal, S. Kalyanaraman, T. Ganu, and M. Grant, "Micro-climate prediction - multi scale encoder-decoder based deep learning framework," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD '21, Virtual Event, Singapore: Association for Computing Machinery, 2021, pp. 3128–3138, ISBN: 9781450383325. DOI: 10.1145/3447548.3467173. [Online]. Available: https://doi.org/10.1145/3447548.3467173.

[17] M. Zanchi, S. Zapperi, and C. A. La Porta, "Harnessing deep learning to forecast local microclimate using global climate data," *Scientific Reports*, vol. 13, no. 1, p. 21 062, 2023.

[18] M. K. Abdelmadjid, S. Noureddine, B. Amina, and B. Khelifa, "Enhancing accuracy in greenhouse microclimate forecasting through a hybrid long short-term memory light gradient boosting machine ensemble approach," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 15, no. 2, pp. 2392–2403, 2025.

[19] L. Pounder. "Raspberry pi produced 10 million rp2040s in 2021, more pi stores likely." Accessed: 02/07/25. [Online]. Available: https://www.tomshardware.com/news/raspberry-pi-10-million-rp2040s.

[20] A. Q. Khan, M. Riaz, and A. Bilal, "Various types of antenna with respect to their applications: A review," *International Journal of Multidisciplinary Sciences and Engineering*, vol. 7, no. 3, pp. 1–8, Mar. 2016, ISSN: 2045-7057. [Online]. Available: https://www.ijmse.org/Volume7/Issue3/paper1.pdf.

[21] simpulse, *Understanding the link budget in wireless communication*, 2025. [Online]. Available: https://www.simpulse-sdr.com/articles/understanding-the-link-budget-in-wireless-communication.

[22] "Solar power management module, for 6v 24v solar panel," Waveshare, Accessed: Aug. 17, 2025. [Online]. Available: https://www.waveshare.com/solar-power-manager.htm.

[23] R. Cathcart. "Best Solar Panel Angle For Maximum Efficiency." Accessed 30 August 2025, SolarFast. [Online]. Available: https://solarfast.co.uk/blog/best-solar-panel-angle/.

[24] Wikipedia contributors, *IP code*, [Online: accessed 11-August-2025]. [Online]. Available: https://en.wikipedia.org/wiki/IP_code.

[25] micromet, *Waterproofing a Capacitance Soil Moisture Sensor*, [Online; accessed 11-Aug-2025]. [Online]. Available: https://www.instructables.com/Waterproofing-a-Capacitance-Soil-Moisture-Sensor/.

[26] D. Halbert, *CircuitPython Safe Mode*, 2023. [Online]. Available: https://learn.adafruit.com/circuitpython-safe-mode/safemode-py.

[27] J. Needell. "Using circuitpython on the challenger rp2040 lora to send and receive data packets," Accessed: Aug. 30, 2025. [Online]. Available: https://ilabs.se/using-circuitpython-on-the-challenger-rp2040-lora-to-send-and-receive-data-packets/.

[28] DFRobot. "Rs485 wind speed meter/sensor/transmitter wiki." Accessed 31 August 2025. [Online]. Available: https://wiki.dfrobot.com/RS485_Wind_Speed_Transmitter_SKU_SEN0483.

[29] "Ir 2030 – uk interface requirements 2030: Licence exempt short range devices (srds)," Ofcom, Tech. Rep., 2023, Accessed: 2025-09-02. [Online]. Available: https://www.ofcom.org.uk/siteassets/resources/documents/spectrum/interface-requirements/ir-2030.pdf?v=335258.

[30] Tailscale. "Tailscale — secure zero-trust networking." Accessed 30 August 2025, Accessed: Aug. 30, 2025. [Online]. Available: https://tailscale.com/.

[31] "Render: Cloud application hosting," Render, Accessed: Aug. 31, 2025. [Online]. Available: https://render.com.

[32] "Typescript: Javascript with syntax for types," Microsoft, Accessed: Aug. 31, 2025. [Online]. Available: https://www.typescriptlang.org.

[33] "Node.js javascript runtime," OpenJS Foundation, Accessed: Aug. 31, 2025. [Online]. Available: https://nodejs.org/en.

[34] "Express: Fast, unopinionated, minimalist web framework for node.js," Express, Accessed: Aug. 31, 2025. [Online]. Available: https://expressjs.com.

[35] B. Carlson. "Pg-pool documentation," Accessed: Aug. 31, 2025. [Online]. Available: https://nodejs.org/en.

[36] M. B. Jones and D. Hardt, *The oauth 2.0 authorization framework: Bearer token usage*, Internet Requests for Comments, RFC, 2012. DOI: 10.17487/RFC6750. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6750.txt.

[37] OpenWeather. "One call api 3.0." Accessed 31 August. [Online]. Available: https://openweathermap.org/api/one-call-3.

[38] "Node-cron (npm package)," npm, Accessed: Aug. 31, 2025. [Online]. Available: https://www.npmjs.com/package/node-cron.

[39] S. Saha. "Xgboost vs lightgbm: How are they different." Accessed: 2025-08-27. [Online]. Available: https://neptune.ai/blog/xgboost-vs-lightgbm.

[40] B. Tuychiev. "A guide to the gradient boosting algorithm." Accessed: 2025-08-29. [Online]. Available: https://www.datacamp.com/tutorial/guide-to-the-gradient-boosting-algorithm.

[41] U. of Reading. "Summer 2025 set to be hottest on university records." Accessed: 2025-08-28. [Online]. Available: https://www.reading.ac.uk/news/2025/Expert-Comment/Summer-2025-set-to-be-hottest-on-University-records.

[42] S. (Seeed), *Sensecap s2120 8-in-1 lorawan weather sensor*, Accessed: 2025-08-29, 2025. [Online]. Available: https://thepihut.com/products/sensecap-s2120-8-in-1-lorawan-weather-sensor?variant=54458635026817.

[43] DecentLab, *Decentlab eleven-parameter weather station for lorawan*, Accessed: 2025-08-29, 2025. [Online]. Available: https://www.alliot.co.uk/product/decentlab-eleven-parameter-weather-station-for-lorawan-special-order-part/.

[44] O. /. HOBO, *Rx3000 system kit (international) - weather station bundle*, Accessed: 2025-08-29, 2025. [Online]. Available: https://www.weathershop.co.uk/rx3000-sys-kit-int-bundle?srsltid=AfmBOoqQWkymjf1HSUGLsR57EoHbqsLn4VCQ00kKK3RauGQ86hw9iMnYyKQ.

[45] S. Electronics, *Sparkfun arduino iot weather station*, Accessed: 2025-08-29, 2025. [Online]. Available: https://thepihut.com/products/sparkfun-arduino-iot-weather-station?srsltid=AfmBOoq78B0tnTz9317fpZFt5icrOpb5GalxbQm7fiBYiqPLlTf5hf3H.

[46] J. Brooke, "Sus: A quick and dirty usability scale," *Usability Eval. Ind.*, vol. 189, Nov. 1995.

[47] J. Sauro and J. R. Lewis, *Quantifying the user experience: Practical statistics for user research*. Morgan Kaufmann, 2016.

[48] Z. Bobbitt, *Mann–Whitney U Test*, Accessed August 30, 2025, 2022. [Online]. Available: https://www.statology.org/mann-whitney-u-test/.

[49] S. S. Statistics, *Mann-whitney u test calculator*, Accessed: 2025-08-29. [Online]. Available: https://www.socscistatistics.com/tests/mannwhitney.

[50] PeterStatistics. "2a: Test for median (one-sample wilcoxon signed-rank test)." Accessed: 30 August 2025. [Online]. Available: https://peterstatistics.com/CrashCourse/2-SingleVar/Ordinal/Ordinal-2a-Test.html.

[51] Stats.Blue. "The wilcoxon signed rank test." Accessed: 30 August 2025. [Online]. Available: https://stats.blue/Stats_Suite/wilcoxon_signed_rank_test.html.

[52] V. Electric, *How lora modulation really works - long range communication using chirps*, Accessed: 2025-08-18, Jul. 2021. [Online]. Available: https://www.youtube.com/watch?v=jHWepP1ZWTk.

[53] L. Vangelista, "Frequency shift chirp modulation: The lora modulation," *IEEE Signal Processing Letters*, vol. 24, no. 12, pp. 1818–1821, 2017. DOI: 10.1109/LSP.2017.2762960.

[54] The Economist, "Solar is going to be huge," *The Economist*, 2024, Accessed: 2025-08-18. [Online]. Available: https://archive.is/W5rHS.

Part V

Appendices

A Breakdown of costs

Category	Name	Units	Cost (£)	Total cost (£)	Units per component				
					Sensor node 1	Sensor node 2	Repeater	Gateway	Difference
Electronics	Electronics iLabs Challenger RP2040 LoRa (868MHz)	4	20.80	83.20	1	1	1	1	-
Electronics	Electronics Solar Power Management Module for 6V-24V Solar Panel	3	9.70	29.10	1	1	1	-	-
Electronics	RS485 Wind Speed Transmitter	2	43.20	86.40	1	1	-	-	-
Electronics	Capacitive Soil Moisture Sensor	2	4.00	8.00	1	1	-	-	-
Electronics	Monocrystalline Silicon Solar Panel	3	9.90	29.70	1	1	1	-	-
Electronics	Electronics 2-Channel RS485 Module for Raspberry Pi Pico	2	7.50	15.00	1	1	-	-	-
Electronics	Electronics 9V Step-Up Voltage Regulator U3V40F9	2	9.60	19.20	1	1	-	-	-
Electronics	18650 Lithium-ion Rechargeable Cell - 2500mAh 3.7V 30A	5	3.99	19.95	2	2	1	-	-
Electronics	Raspberry Pi	1	33.60	33.60	-	-	-	1	-
Electronics	iLabs LoRa Antenna (EU868)	4	5.53	22.12	1	1	1	1	-
Electronics	Electronics DHT11 Temperature-Humidity Sensor	2	3.90	7.80	1	1		-	-
Electronics	Electronics Half size breadboards	3	2.00	6.00	1	1	1	-	-
Electronics	Electronics 5 Meters of Black 3 core cable	1	5.00	5.00	0.50	0.50	-	-	-
Electronics	Electronics Battery storage case	5	1.00	5.00	2	2	1	-	-
Electronics	Assorted (M-M, M-F, F-F) jumper cables	1	6.00	6.00	0.25	0.25	0.25	0.25	-
Enclosure	IP65 junction box	3	12.18	36.54	1	1	1	-	-
Enclosure	Small IP55 junction box	2	4.20	8.40	1	1	-	-	-
Enclosure	Wooden posts (2m)	3	17.00	51.00	1	1	1	-	-
Enclosure	Gravel board	1	8.50	8.50	0.5	0.5	-	-	-
Enclosure	Plastic box with screw on lid	1	5.00	5.00	-	-		1	-
Enclosure	Steel spur post supports	3	8.95	26.85	1	1	1	-	-
				512.36	176.69	176.69	92.55	66.43	-

Figure 38: Table of component costs

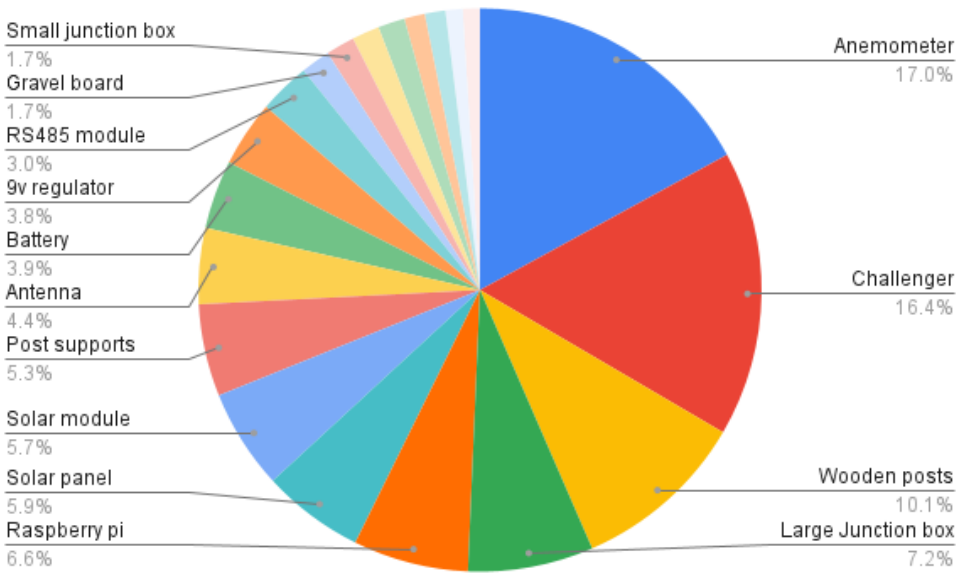


Figure 39: Chart to visualise relative cost of components

B Interview excerpt with Small Brook Farms owners

Interviewer: So you would need a weather station to observe very local weather, I expect. What you're saying is that the BBC website weather is not necessarily relevant to you?

Speaker 1: No, No , so there's another site which is in Sanford and we can have conversations - we're what? - 5 miles apart 10 miles?

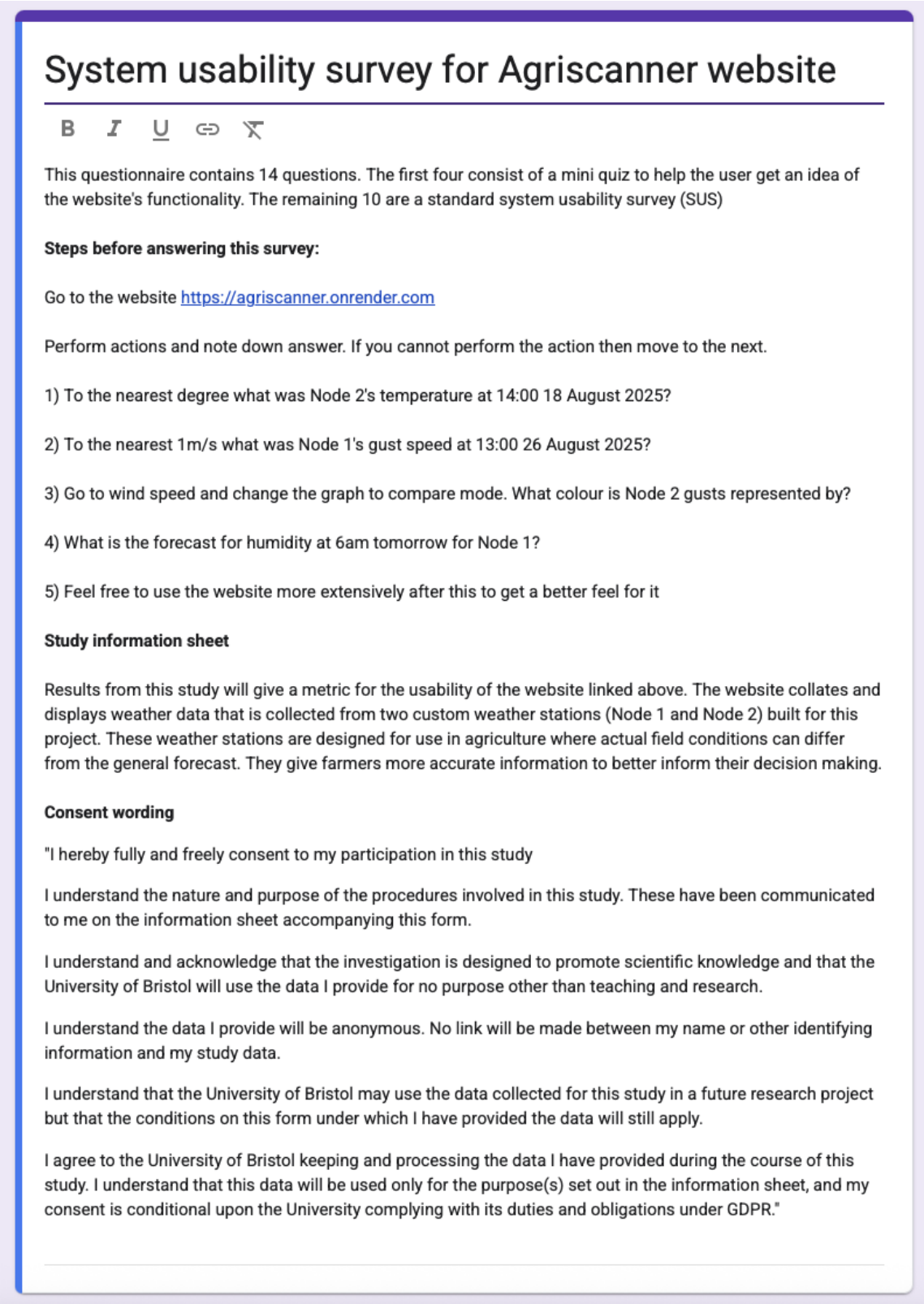
Speaker 2: Yeah. So the other side of that hill there like a mile away you get a different sort of weather, but even on this side of the [apple orchard] as opposed to that side of the [apple orchard] like the wind can be less than whatever else, its very localised. But you know you if you then go on that side of the valley, it doesn't rain on this side. So like to be actually useful, yeah, [weather monitoring] sort of has to be [based on] the farm.

Source: Transcript no.28 of site visit from DECIDE sharepoint (9 May 2025)

C Battery cost assumptions

- **Agriscanner Network:** Replace 5 (2 per sensor node, 1 for repeater) Li-ion batteries every 3 years at a cost of £20 $\Rightarrow \approx$ £7 per annum.
- **SenseCAP S2120:** Three AA batteries per node, replaced twice a year (total 12 batteries) $\approx$ £10 per annum.
- **Decentlab Eleven Parameter:** Two C batteries per node, replaced four times per year (total 16 batteries) $\approx$ £30 per annum.
- **HOBO weather station kit:** Replace lead-acid for each node battery every 4 years at a cost of £80 $\Rightarrow \approx$ £20 per annum.

D System usability survey



System usability survey for Agriscanner website

This questionnaire contains 14 questions. The first four consist of a mini quiz to help the user get an idea of the website's functionality. The remaining 10 are a standard system usability survey (SUS)

Steps before answering this survey:

Go to the website https://agriscanner.onrender.com

Perform actions and note down answer. If you cannot perform the action then move to the next.

1) To the nearest degree what was Node 2's temperature at 14:00 18 August 2025?

2) To the nearest 1m/s what was Node 1's gust speed at 13:00 26 August 2025?

3) Go to wind speed and change the graph to compare mode. What colour is Node 2 gusts represented by?

4) What is the forecast for humidity at 6am tomorrow for Node 1?

5) Feel free to use the website more extensively after this to get a better feel for it

Study information sheet

Results from this study will give a metric for the usability of the website linked above. The website collates and displays weather data that is collected from two custom weather stations (Node 1 and Node 2) built for this project. These weather stations are designed for use in agriculture where actual field conditions can differ from the general forecast. They give farmers more accurate information to better inform their decision making.

Consent wording

"I hereby fully and freely consent to my participation in this study

I understand the nature and purpose of the procedures involved in this study. These have been communicated to me on the information sheet accompanying this form.

I understand and acknowledge that the investigation is designed to promote scientific knowledge and that the University of Bristol will use the data I provide for no purpose other than teaching and research.

I understand the data I provide will be anonymous. No link will be made between my name or other identifying information and my study data.

I understand that the University of Bristol may use the data collected for this study in a future research project but that the conditions on this form under which I have provided the data will still apply.

I agree to the University of Bristol keeping and processing the data I have provided during the course of this study. I understand that this data will be used only for the purpose(s) set out in the information sheet, and my consent is conditional upon the University complying with its duties and obligations under GDPR."

Figure 40: SUS survey wording

List of tasks for users:

1. To the nearest degree, what was Node 2's temperature at 14:00 18 August 2025?

2. To the nearest 1 m/s, what was Node 1's gust speed at 13:00 26 August 2025?
3. Go to wind speed and change the graph to compare mode. What colour is Node 2 gusts represented by?
4. What is the forecast for humidity at 06:00 tomorrow for Node 1?

E Additional SUS materials

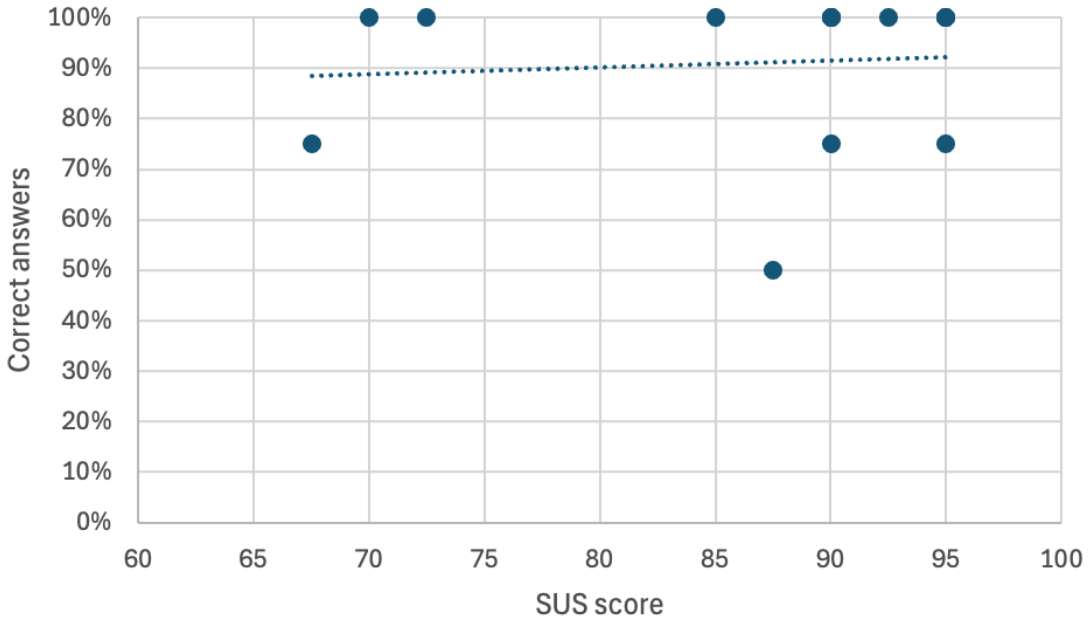


Figure 41: Graph to show SUS score vs percent of correct answers ($R^2 = 0.0066$)

responder_num	sus_score	correct_answers	type
1	90.0	75%	desktop
2	92.5	100%	mobile
3	95.0	100%	mobile
4	70.0	100%	desktop
5	90.0	100%	mobile
6	90.0	100%	desktop
7	87.5	50%	desktop
8	95.0	100%	mobile
9	95.0	75%	mobile
10	95.0	100%	desktop
11	72.5	100%	desktop
12	67.5	75%	mobile
13	100.0	100%	mobile
14	90.0	100%	mobile
15	85.0	100%	desktop

Table 8: Raw SUS scores and correct answer percentage from survey

“I found it very hard to get to a specific time on the graph - the granularity of the cursor moving seemed to make it hard to choose my time - I ended up with 14:01 for the first question as I couldn’t get the cursor to stay at 14:00. But as a weather nerd I loved it.”

“I found it annoying having to click to get to the required date. Also I don’t understand from the website alone what the project is about or where the data is from - maybe an about page would be nice :)”

“Sorry wasn’t sure where the compare graph is but the website looked really nice!”

Figure 42: Feedback from participants with lower SUS scores

"The soil moisture readings surprised me - I expected them to go up with rain (increased moisture) but they went down. I think this is counter-intuitive and there should either be some explanation of the what the measurements mean, or preferably there should be an option to convert to a more human understandable description like very dry/dry/slightly moist.../wet/very wet/saturated"

"When selecting the date (specifically when clicking the arrows to move forwards and backwards in time) it would be useful to have a calendar display to travel to past dates quicker."

"Extra features: - Date picker instead of having to navigate past each day - Export feature to export data in bulk (e.g. to a csv) - Ability to select a time period (specific dates) instead of just a day view"

"I think when switching dates, it should support selecting a date from the calendar rather than only moving forward or backward to the nearest dates."

"Being able to navigate directly between measurements (e.g. temp, humidity etc.) while on [sic]"

"Finding the "compare" option was the hardest part. But didn’t take long."

"Great website- simple layout and easy to use!"

"No bugs seen, but on wind graph some of the y axis text was slightly cut off on my screen. I’m on a laptop"

Figure 43: Feedback from participants with higher SUS scores

Both the Mann Whitney U and Wilcoxon Signed Rank test were selected for this data because the SUS score is derived from Likert items. Likert items are ordinal (e.g. strongly disagree) and so nonparametric statistical testing is typically preferred [48]

The Mann Whitney U statistical test score was calculated in an Excel sheet I made using the procedure described in [48] and the result cross-verified using the website in [49] which is also the source of the p-value. Results: U = 13.5, critical value = 10 p=0.10524. Parameters: two-tailed, 0.05 significance

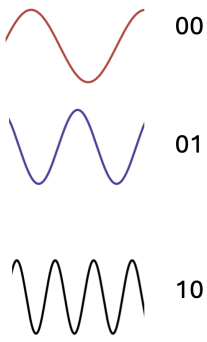
Due to the complexity of calculating it, the one-sample Wilcoxon Signed Rank Test score was calculated in Excel using a modified template from the source in [50] and cross-verified with the website in [51]. Results: $W^+ = 119$ z = 3.3361, critical value = 1.6449 Parameters: right-tailed, 0.05 significance

Figure 44: Note on statistical testing

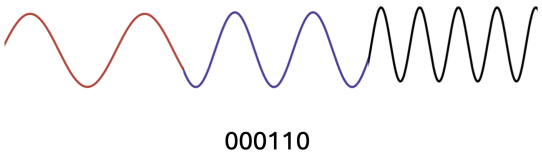
F How LoRa works

As this paper is not a technical study of radio communication I will opt for a brief summary of the principles behind LoRa. With this in mind I have based much of the information from the excellent video lecture in [52] that itself draws upon the paper in [53].

The reason LoRa modulation is different to traditional modulation techniques is the use of a "chirp" as the key to transmitting packets. A more traditional technique might involve a frequency shift key; that is a single frequency represents several bits. These unique frequencies are called symbols as they represent data, like letters in the alphabet. In the below graph we see three simplified symbols that represent binary values, the combination of these symbols makes a packet:



(a) Frequency shift symbols



(b) Packet

Figure 45: Traditional radio modulation with frequency symbols

In traditional modulation, symbols always have flat unchanging frequency (as can be seen from the fact the wave separation never changes). LoRa symbols instead have changing frequencies that have a waveform like the below.

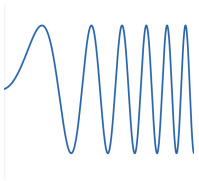
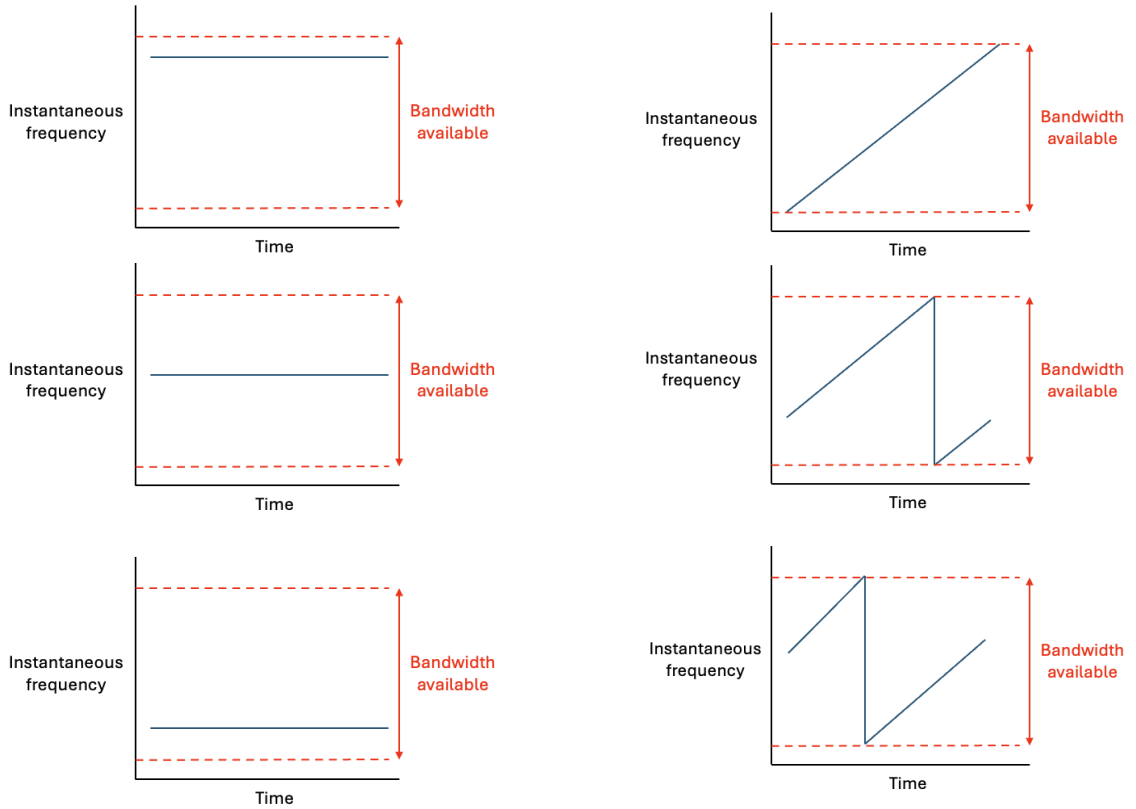


Figure 46: LoRa symbol showing changing frequency ("Up-chirp")

This change in frequency is what gives the wave form the name "chirp". If traditional frequency symbols were thought of a sound they would be similar to morse code beeps while LoRa would be more similar to siren or *chirp*ing bird. LoRa has both rising and falling chirps (up-chirps and down-chirps).

Different LoRa symbols are then distinguished by the point in time of a discontinuity. LoRa symbols are always delivered over a known length of time, so different symbols include a reset back to the starting frequency at a different point in time.

A way to graphically show this discontinuity in LoRa and compare it to frequency shift modulation is by using instantaneous frequency graphs:



(a) Frequency shift symbols

(b) LoRa symbols, note shifting discontinuity

Figure 47: Comparison of frequency shift and LoRa modulation

Once these symbols hit the receiver, the receiver must work out which symbol it was. In frequency shift modulation this is achieved by performing a correlation test against every symbol received. However this is computationally difficult and requires a low signal-to-noise ratio to work effectively. The benefit of chirps is that due to a mathematical transformation that will not be further discussed (Fast Fourier transform), the correlation can be computed much more easily and with less powerful

hardware.

G IoT enabling technologies

This supplementary section lists the key technological developments that have are contributed to the viability of my project.

1. Efficiency improvements in microchips - breakthroughs in microchip fabrication have led to smaller more efficient chips with improved performance.
2. Lithium-Ion batteries improvements - continuous improvements in the energy density of lithium-ion batteries has made it possible to power devices for long periods without mains power.
3. Low-power long-range radio - new radio communication techniques such as LoRa allow for data transmission over several kilometres using a fraction of the power required by traditional mobile or Wi-Fi technologies.
4. Affordability of solar panels - since 1970 the price of solar panels has decreased to 1/500th of its original cost [54] making solar a viable power source for IoT systems.
5. Growth of hobbyist embedded systems - since the release of accessible platforms such as Arduino in 2005, the growth of hobby level embedded systems has lowered the barrier to entry to create IoT systems.
6. Accessible cloud computing and hosting - Cheap and available web hosting has allowed application level systems to be more easily developed.

H CircuitPython sensor node 1 code example

```
#Code adapted from iLabs example by Jerry
    Needle 2021 (the LoRa setup)

import time
import board
import busio
import digitalio
import adafruit_rfm9x
import adafruit_dht
import analogio


DEVICE_ID = 1
RADIO_FREQ_MHZ = 868.0
WIND_REQUEST = bytes([0x02, 0x03, 0x00, 0x00,
     0x00, 0x01, 0x84, 0x39])

# Initialise LORA radio and settings
try:
    spi = busio.SPI(board.RFM95W_SCK, MOSI=
        board.RFM95W_SDO, MISO=board.
        RFM95W_SDI)
    cs  = digitalio.DigitalInOut(board.
        RFM95W_CS)
    rst = digitalio.DigitalInOut(board.
        RFM95W_RST)
    rfm9x = adafruit_rfm9x.RFM9x(spi, cs, rst
        , RADIO_FREQ_MHZ)

    rfm9x.tx_power = 13
    rfm9x.spreading_factor = 7
    rfm9x.signal_bandwidth = 125_000
    rfm9x.coding_rate      = 5
    rfm9x.enable_crc       = True
    rfm9x.implicit         = False

except Exception as e:
    rfm9x = None
    print("ERR: Lora module", e)

# Initialise UART and settings
try:
    uart = busio.UART(tx=board.GP16, rx=board
        .GP17, baudrate=9600, timeout=0.2)
except Exception as e:
    uart = None
    print("ERR: UART", e)

# Initialise the soil moisture sensor
try:
    moisture_pin = analogio.AnalogIn(board.A0
        )
except Exception as e:
    moisture_pin = None
    print("ERR: moisture pin", e)

# Initialise DHT11 temperature humidity
    sensor
try:
    dht_device = adafruit_dht.DHT11(board.A1)
except Exception as e:
    dht_device = None
    print("ERR: dht11", e)

# Returns raw soil moisture reading
def get_raw_moisture(pin):
```

```
    try:
        raw = pin.value
        return raw
    except:
        return None

# Returns wind speed sensor using MODBUS
    protocol
def get_wind_speed():
    if uart:
        try:
            uart.write(WIND_REQUEST)
            time.sleep(0.2)
            response = uart.read(16)
            if response:
                if len(response) >= 5 and
                    response[0] == 0x02 and
                    response[1] == 0x03:
                    value_raw = response[3]
                        << 8 | response[4]
                    return value_raw / 10.0
                else:
                    print("Bad format error")
                    return None
            else:
                print("No response error")
                return None
        except:
            print("Unknown error")
            return None
    print("Bad UART error")
    return None

def find_err(value):
    if value is None:
        return "ERR"
    else:
        return value

counter = 0

while True:

    t_sum = h_sum = w_sum = 0.0
    s_sum = 0
    t_n = h_n = s_n = w_n = 0
    min_counter = 0
    w_max = 0

    # Do 10 readings each minute and then
        average the minute result
    # If an exception occurs use pass to keep
         running
    while min_counter < 10:
        if dht_device:
            try:
                t_reading = dht_device.
                    temperature
                if t_reading is not None:
                    t_sum += float(t_reading)
                    t_n += 1
            except Exception:
                pass
            try:
                h_reading = dht_device.
                    humidity
                if h_reading is not None:
                    h_sum += float(h_reading)
                    h_n += 1
            except Exception:
                pass
        s_reading = get_raw_moisture(
            moisture_pin)

        if s_reading is not None:
            s_sum += s_reading
            s_n += 1

        w_reading = get_wind_speed()

        if w_reading is not None:
            w_sum += float(w_reading)
            w_n += 1
            if w_reading > w_max:
                w_max = w_reading
        print(f"{t_reading}, {h_reading}, {
            s_reading}, {w_reading}")
        min_counter += 1
        time.sleep(6)


if t_n != 0:
    t_avg = round(t_sum/t_n, 1)
else:
    t_avg = None
if h_n != 0:
    h_avg = int(round(h_sum/h_n, 1))
else:
    h_avg = None
if s_n != 0:
    s_avg = int(round(s_sum/s_n, 0))
else:
    s_avg = None
if w_n != 0:
    w_avg = round(w_sum/w_n, 1)
else:
    w_avg = None
    w_max = None

#send payload and include errors if found
payload = f"{DEVICE_ID},{find_err(t_avg)
    },{find_err(h_avg)},{find_err(s_avg)
    },{find_err(w_avg)}, {find_err(w_max)
    },{counter}"
try:
    rfm9x.send(payload.encode("utf-8"))
    print(f"Sent packet {payload}")
#If sending the payload fails then print
    error
except Exception as e:
    print("Failed to send packet: ", e)

counter += 1
```

I Typescript API endpoint code for node data insert

```
// End point to insert into node_data table
app.post('/api/database/insert-node-data',
    async (req: Request, res: Response) => {
  //Assign JSON body variables
    const {
      device_id,
      packet_id,
      temperature,
```

```
    humidity,
    soil_moisture,
    wind_speed,
    gust_speed,
    rssi0,
    rssi1,
    snr0,
    snr1,
  } = req.body;

  //attempt an insert with values from the
      json body
  try {
    const query = `INSERT INTO node_data
      (node_deployment_id, farm_id, packet_id
          , temperature, humidity,
      soil_moisture, wind_speed, gust_speed,
          rssi0, rssi1, snr0, snr1, node_name
          )
      VALUES (
      (SELECT id FROM node_deployment WHERE
          node_name = $11 ORDER BY ts DESC
          LIMIT 1),
      (SELECT farm_id FROM node_deployment
          WHERE node_name = $11 ORDER BY ts
          DESC LIMIT 1),
        $1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11);`;
    const values = [
      packet_id,
      temperature,
      humidity,
      soil_moisture,
      wind_speed,
      gust_speed,
      rssi0,
      rssi1,
      snr0,
      snr1,
      device_id
    ];
    //put query into pooled connection
    await pool.query(query, values);
    //return success if successful
    res.status(200).json({ status: 'success'
        });
  } catch (error) {
    console.error(error);
    //return error if error occurs in insert
    res.status(500).json({ error: 'failed to
        insert to database' });
  }
});
```

J Python code to train machine learning model

```
import pandas as pd
import numpy as np
import lightgbm as lgb
import joblib as jl
import m2cgen as m2c
from sklearn.metrics import
    mean_absolute_error, mean_squared_error

VALIDATION_PERCENT = 0.2
NODE_ID = 1
TARGET = 'temperature'

NODE_PREFIX = 'node_' + f'{NODE_ID}'
TARGET_COL = NODE_PREFIX + '_' + TARGET

# Read the csv and make it into pandas
    dataframe
data_frame = pd.DataFrame(pd.read_csv(
    NODE_PREFIX + '_training_data.csv'))

# Remove sensor data other than the target
cols_to_remove = ['ts']
for col in data_frame.columns:
    col_name = str(col)
    if col_name != TARGET_COL and col_name.
        startswith(NODE_PREFIX):
        cols_to_remove.append(col_name)

data_frame = data_frame.drop(columns =
    cols_to_remove)

# Set the split for training and validation
    data
cut_off_index = int(round((1-
    VALIDATION_PERCENT) * len(data_frame), 0)
    )

# Define the data
training_data = data_frame.iloc[:
    cut_off_index]
validation_data = data_frame.iloc[
    cut_off_index:]

# Define the features used for prediction
features = ['day_sin', 'day_cos', 'year_sin',
     'year_cos', 'temp', 'pressure',
'humidity', 'uvi', 'clouds', 'wind_speed', '
    wind_gust', 'rain_1h', 'snow_1h']

# Initialise LGBM with following parameters
training_model = lgb.LGBMRegressor(
    n_estimators=250,
    learning_rate=0.05
)

# Fit the model using features and target
    data
training_model.fit(
    training_data[features],
    training_data[TARGET_COL],
    eval_metric='rmse',
    eval_set=[(validation_data[features],
        validation_data[TARGET_COL])],
    # Stop running training after 50
        iterations of no improvement
    callbacks=[lgb.early_stopping(50), lgb.
        log_evaluation(20)]
)

# If it ends from best iteration get the date
     from this
best_iteration = getattr(training_model, "
    best_iteration_", None)
if best_iteration:
    prediction = training_model.predict(
        validation_data[features],
        num_iteration=best_iteration)
else:
```

```
    training_model.predict(validation_data[
        features])

#Console log useful stats
print("Mean absolute error:",
    mean_absolute_error(validation_data[
    TARGET_COL], prediction))
print("Root mean squared error:", np.sqrt(
    mean_squared_error(validation_data[
    TARGET_COL], prediction)))
```

```
print("Features used:", training_model.
    booster_.feature_name())

#Export final model to JS
javascript_final_model = m2c.
    export_to_javascript(training_model)

with open(f'{TARGET}{NODE_ID}.js', 'w') as f:
    f.write(javascript_final_model)
```

K Mean absolute error (MAE) and root mean squared error (RMSE) from forecast comparison

	Node 1					Node 2				
	temperature	humidity	wind speed	gust speed	soil moisture	temperature	humidity	wind speed	gust speed	soil moisture
Observed Average	16.23	84.29	1.38	2.29	24285.56	16.24	84.31	1.29	2.18	21819.56
MAE general forecast	0.56	4.65	4.40	8.48	N/A	0.56	4.67	4.48	8.59	N/A
Relative MAE (%)	3%	6%	320%	370%	N/A	3%	6%	347%	393%	N/A
MAE ML prediction	0.72	10.75	0.71	1.01	24945.52	0.21	14.33	0.46	0.68	48297.54
Relative MAE (%)	4%	13%	52%	44%	103%	1%	17%	35%	31%	221%
Alternative prediction	0.12	9.04	3.07	5.88	N/A	0.29	9.62	3.11	5.92	N/A
Relative MAE (%)	1%	11%	224%	257%	N/A	2%	11%	241%	271%	N/A
RMSE general forecast	1.61	12.56	4.60	8.85	N/A	1.56	12.19	4.69	8.97	N/A
Relative RMSE (%)	10%	15%	335%	387%	N/A	10%	14%	363%	411%	N/A
RMSE ML prediction	2.08	12.76	1.10	1.40	24973.58	2.00	18.30	0.46	0.72	48298.05
Relative RMSE (%)	13%	15%	80%	61%	103%	12%	22%	36%	33%	221%
RMSE Alternative prediction	1.51	14.76	3.35	6.40	N/A	1.48	14.81	3.40	6.45	N/A
Relative RMSE (%)	9%	18%	244%	280%	N/A	9%	18%	264%	296%	N/A

Figure 48: General forecast is OpenWeather, Machine learning refers to my models, alternative refers to a mean adjusted general forecast model

	Average MAE (%)					Average RMSE (%)				
	temperature	humidity	wind speed	gust speed	soil moisture	temperature	humidity	wind speed	gust speed	soil moisture
General forecast	3.4%	5.5%	333.6%	381.9%	N/A	9.8%	14.7%	348.8%	398.6%	N/A
ML prediction	2.9%	14.9%	43.6%	37.6%	162.0%	13%	18%	58%	47%	162%
Alternative prediction	1.3%	11.1%	232.4%	264.0%	N/A	9.2%	17.5%	253.7%	287.6%	N/A

Figure 49: Table with relative MAE and RMSE averaged across node 1 and 2

L Alternative model data

The below figure was used to produce the alternative model by applying these adjustments to the general OpenWeather forecast. A positive reading means sensor data was higher than forecast.

	temp	humidity	wind_speed	wind_gust
node_1	0.67	-4.40	-1.33	-2.60
node_2	0.85	-4.95	-1.37	-2.67

Figure 50: Spreadsheet snippet showing average difference between general forecast and sensor readings between (15-27 August)