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: TBD

Executive summary

Dedication and acknowledgments

Author's declaration

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 separate postgreSQL database that collects and stores readings from my weather station sensor nodes every minute. These 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 consisted 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 designed for deployment on Small Brook Farm, an apple orchard who primarily sell their produce to cider makers. Unfortunately, actual deployment at this location was not possible during the time frame so the evaluation section of this report is focused on data collected while the weather station network was set up in my garden.

At the time of submission, discussions with the owner of Small Brook Farm were ongoing to deploy the system in their orchard as part of a funded university project known as DECIDE.

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 single farm or even 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 on 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 even mentioned that the weather conditions across a single field of apple trees was different, with higher winds on one side compared to other.

This uncertainty has real world consequences for farmers, at small brook farm for instance wind speed is a key determinant of when the farmers can spray their crops with pesticides. Equally, general forecasts are not well equipped to predict a spring frost that damages crops or the precise 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 their specific location.

1.2 Contributions

The key contributions of this paper are:

- The development of a low cost, ultra long-range remote weather station system with superior range and reduced cost compared to current solutions on the market.
- A publicly available web application that visualises live data from multiple field locations, with a high degree of usability when evaluated qualitatively using the System Usability Scale (SUS).
- A novel method for enabling microclimate forecasting using a machine learning process that compares local sensor data with broader regional weather forecasts.

1.3 Layout of dissertation

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 2 details the hardware development process, covering the rationale behind component selection, the testing of hardware, and how the IoT nodes were deployed in the field.

Part 3 focuses on the software aspect of the project, outlining the design and implementation of both the backend infrastructure and the frontend web interface.

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.

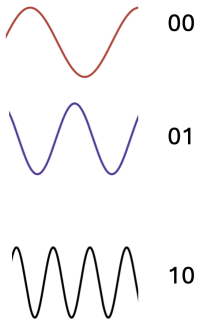
2.1 What is LoRa?

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 widespread than these other networks. Compared to WiFi, LoRa has a range over 4000 times greater [1]. The world record distance for a LoRa signal is 1336km [2]. This was achieved under abnormal conditions: a fishing boat near Portugal with a LoRa transmitter had one of its packets received on the high altitude Canary islands. This exceptional range is also achieved with remarkably little power.

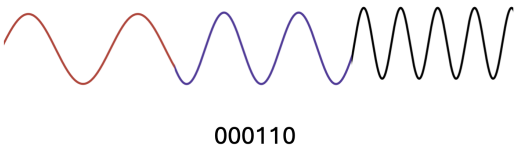
2.2 How LoRa works

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 [3] that itself draws upon the paper in [4].

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 1: 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.

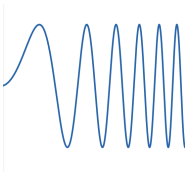
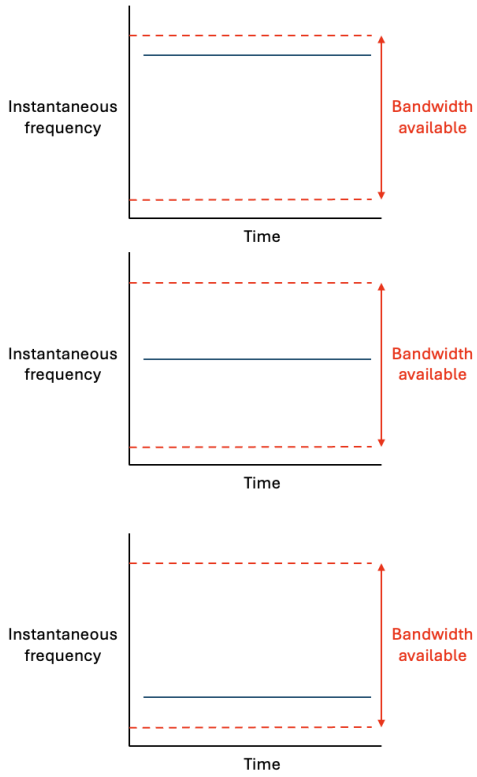


Figure 2: LoRa symbol showing changing frequency ("Up-chirp")

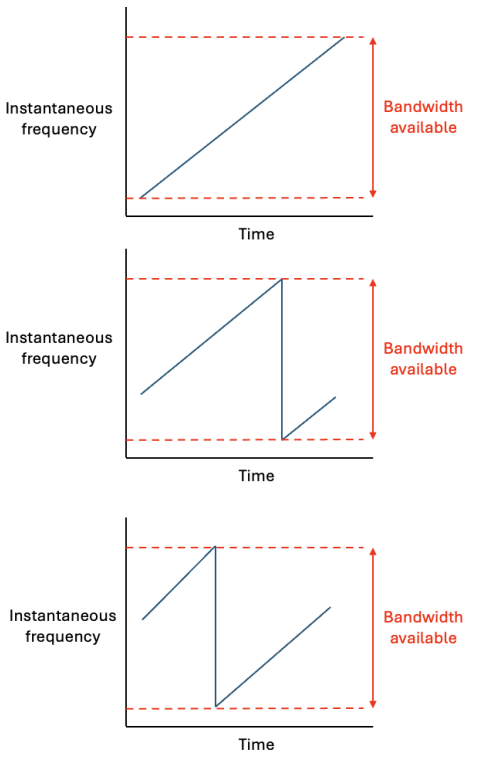
This change in frequency is what gives the wave form the name "chirp". If traditional frequency symbols were thought of a sound they would be similar to morse code beeps while LoRa would be more similar to siren or *chirp*ing bird. LoRa has both rising and falling chirps (up-chirps and down-chirps).

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 3: 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.

2.3 Benefits and limitations

The benefits 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 also demodulate with a lower power budget because of the easier correlation process with LoRa chirps.

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 it's low power.

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: [5])

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 [6]. A similar but more colloquial term would be "smart" devices.

The number of IoT devices is forecasted to reach 40 billion by 2030 from 16.6 billion in late 2023 [7]. IoT devices are already used extensively in a multitude of commercial and domestic settings.

3.2 The layers of IoT

All IoT relies on the existence of three fundamental network layers [8], working from the bottom to the top there is:

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.

3.3 IoT enabling technologies

The surge in IoT for the past few decades has been fuelled by the emergence of new technologies. Here I will explore the technologies that have are most relevant to the weather stations built for this 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 [9] 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.

3.4 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 cut costs by bringing digital solutions that would not have previously been viable without access to power and internet connectivity. 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. [10] 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, mapping and imaging.

IoT weather stations like those developed in this project are a growing trend in agriculture.

3.5 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 which is not viable for most farms as there is a large separation between the fields they need to be deployed in and buildings with WiFi access. As discussed in section 2.3, WiFi has just a fraction of the range of LoRa so these types of stations are not comparable to my own.

For a detailed comparison of different models compared to my own see the relevant section in the evaluation.

4 Microclimates

This section gives 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 [11]. 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 [12]. 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 also trap higher concentrations of pollution from vehicle emissions [13]. 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 [14].

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 [15]. 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 pathogen habitats (such as hedges and cattle fields). The data revealed that the daytime temperature in these microenvironments 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 outbreaks [16].

Another important aspect of microclimates for farmers is how it can affect frost risk - sudden and unpredictable drops below freezing in crop fields that damages 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 of cold air where temperatures can be several degrees lower than the surrounding landscape [17]. These local conditions may not be captured by 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. A 2021 study by Kumar et al developed a machine learning framework that is able to predict a variety of climatic variables such as soil moisture, wind speed and temperature. They were able to get up to 90% accuracy with a 12-120 hour forecast range.

[MUST EXTEND THIS]

can include this for e.g.:

https://ijece.iaescore.com/index.php/IJECE/article/view/36633/18157

https://rmets.onlinelibrary.wiley.com/doi/10.1002/met.2200

below is very close to mine. https://www.nature.com/articles/s41598-023-48028-1

Part II

Hardware development

5 Overview of hardware

The design for the final IoT system involved three distinct module types: Sensor node, repeater and gateway. The system involves two separate sensor nodes that are placed in range of the repeater. The repeater is then placed in range of an internet-enabled gateway endpoint to allow the uploading of climate data to the cloud.

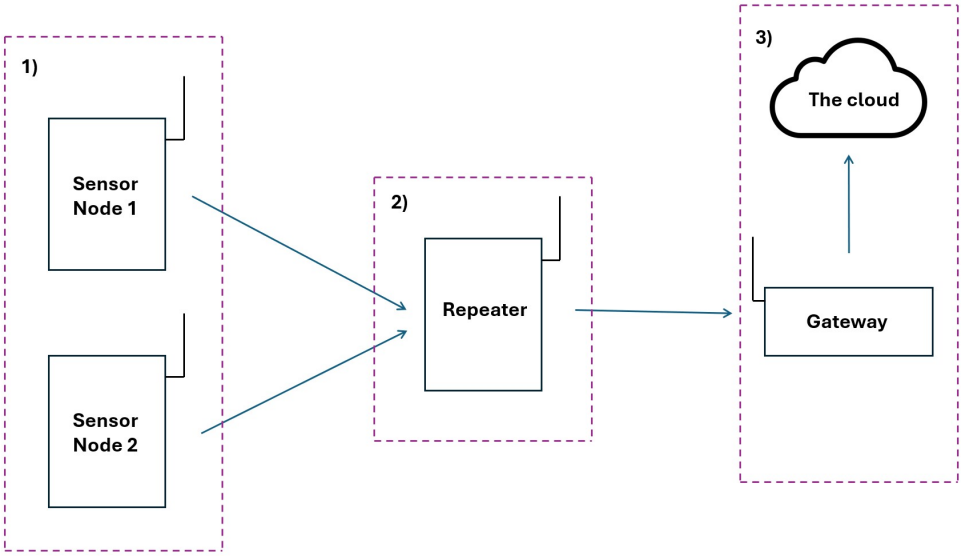


Figure 4: Network diagram of the system

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 repeater is 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

3. Gateway: The final part of the hardware system is the gateway - which is also part of the network layer. The gateway consists of a challenger to receive LoRa signals connected to a raspberry pi which can read the the decoded LoRa messages from the challenger's serial output. The raspberry pi also has a WiFi radio onboard so it is responsible for the uploading of data to the cloud.

6 Design

6.1 Nodes

6.1.1 Components

The first step in the design process was selecting hardware components that could operate autonomously without mains power. This required devices that were highly power-efficient, while also being capable of transmitting small data packets via LoRa. An equally important consideration was weatherproofing the final to protect the sensitive electronics from water ingress and environmental damage.

Challenger RP2040 LoRa

The iLabs Challenger RP2040 LoRa 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 with ample power to perform 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 [18]. This popularity means there is ample documentation for developing with this processor and it is compatible with circuit python which was my preferred language for development.

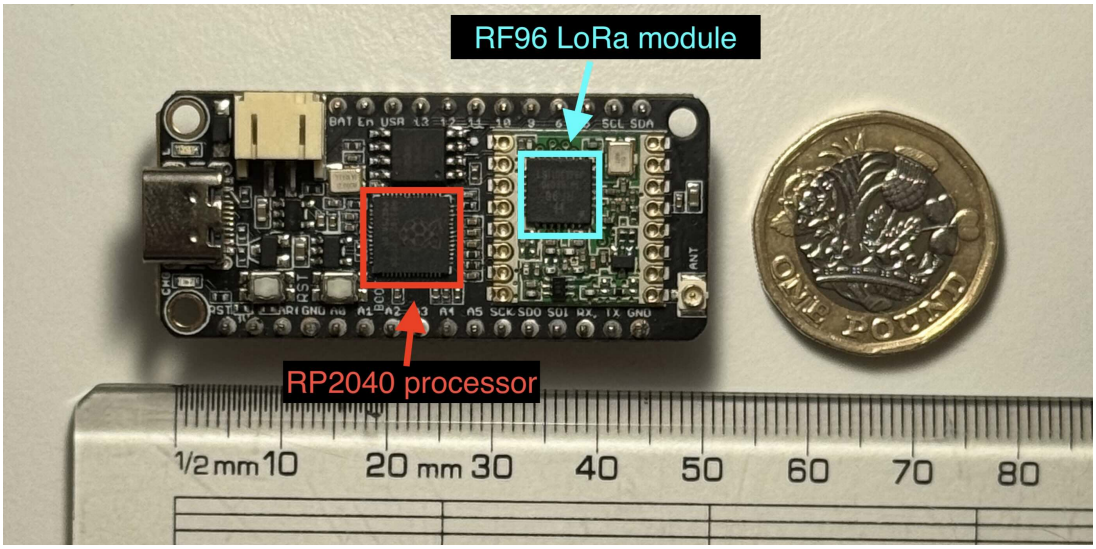


Figure 5: iLabs Challenger RP2040

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.

Antennae

The selection of antennae is one of the largest determinants of range and reliability in the context of wireless communication systems [19]. Initially I used a simple PCB antenna as shown in Figure 6, however as explained in the next chapter, the range of this was insufficient for my use.

Figure 6: 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.

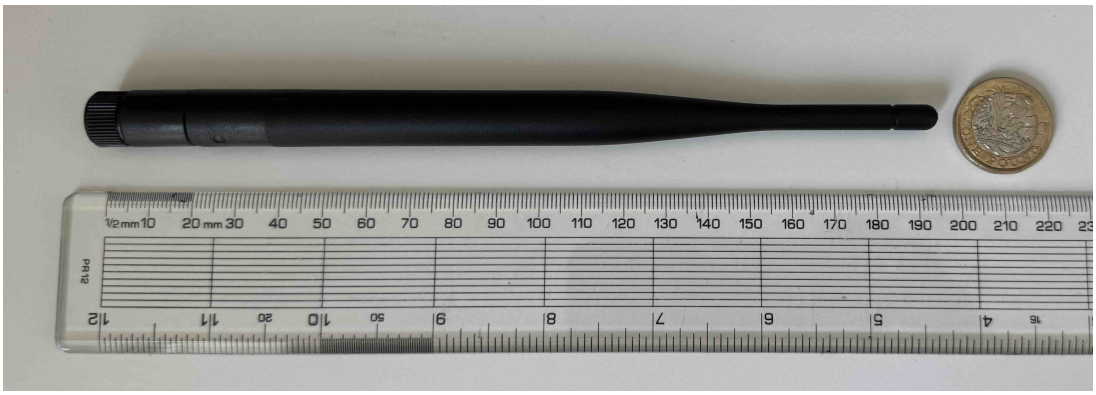


Figure 7: iLabs whip style LoRa antenna (868mhz)

Sensor selection

Temperature and humidity sensor

I used a DHT11 temperature/humidity sensor for each 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.

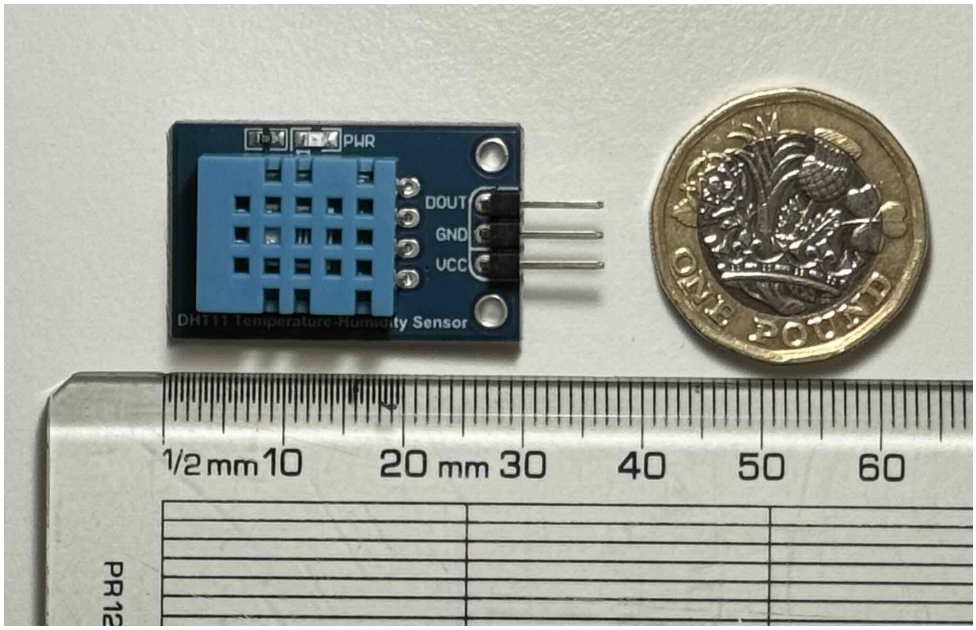


Figure 8: Waveshare DHT11 temperature/humidity sensor

Soil moisture sensor

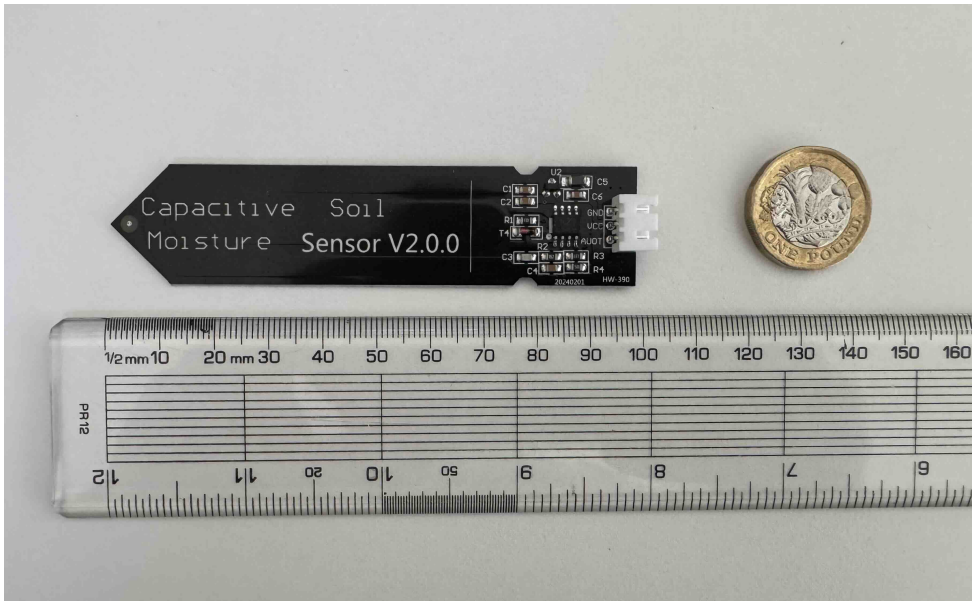


Figure 9: The Pi Hut capacitive soil moisture sensor

A capacitative soil moisture sensor was chosen over the alternative resistive style. The benefit of capacitative sensors is they are not as prone to corrosion of the diodes. 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. For this reason I went with a capacitive sensor as the diodes are covered by a protective layer making them much less susceptible to corrosion.

Wind speed sensor (Anemometer)

Figure 10: DFROBOT wind speed sensor

The anemometer chosen 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. At the time of purchase I was unaware that the RS485 serial communication protocol that the sensor used was not out of the box compatible with the challengers UART protocol. Fortunately, I was able to purchase an RS485 to UART serial converter that allowed the devices to communicate.

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 challengers voltage to the required level.

Powering the node

To allow for continuous operation away from power sources, I set up a 6W Monocrystalline Silicon Solar Panel to each of the nodes. As the output from the solar panels was at too high a voltage to directly power the nodes, and as power was required overnight, I also installed a solar power management module onto the Challengers. This module uses the solar panel to charge a battery that is installed on it. When solar power dips overnight the battery will discharge to power throughout the day.

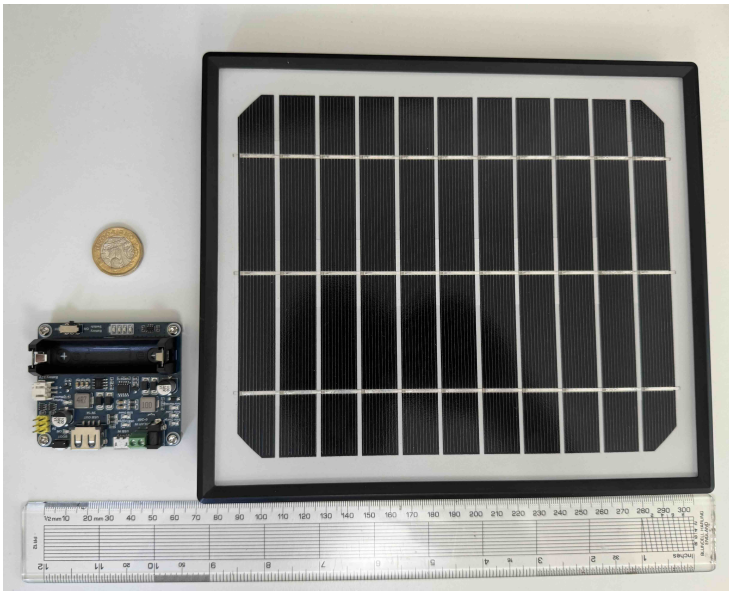


Figure 11: Waveshare solar power management module (left), solar panel (right)

6.1.2 Repeater node

To improve range I used one of the challenger boards as a repeater. This meant it did not require sensors and instead simply had an antenna, solar panel and battery connected up to it. The repeater would receive signals from the two nodes before relaying them to the final gateway. This has the effect of essentially doubling range.

6.2 Gateway

The gateway consisted of a fourth Challenger RP2040 connected to a Raspberry Pi 5 with 8GB of memory. This raspberry pi was significantly more powerful than was actually needed for the purpose of a gateway and a cheaper alternative would have been sufficient however I was able to use a university provided one for the project. The challenger receives messages from the repeater and then transfers these to the Raspberry Pi. After receiving the challengers output the raspberry pi sends an POST response to my API which then inserts the data into a separate SQL database.

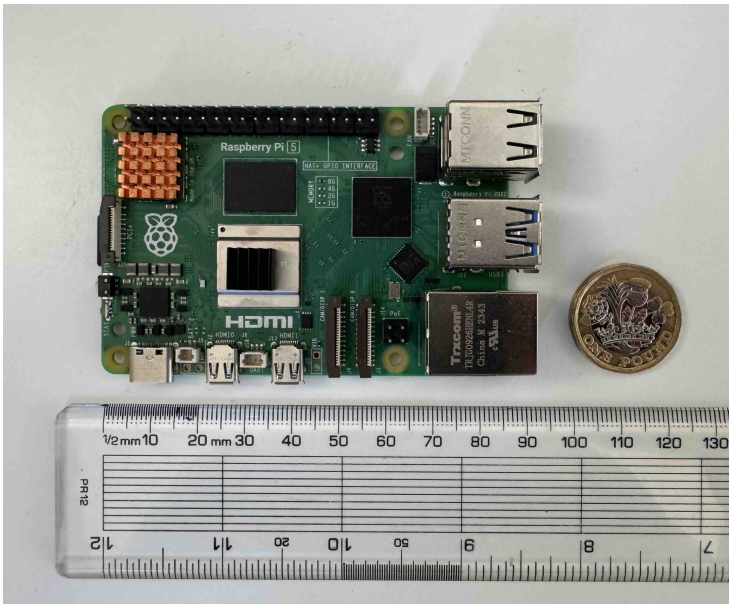


Figure 12: Raspberry Pi 5 (8GB)

7 Development and testing

7.1 Range tests

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

In 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 and measured in decibels.

For the test, two challengers were programmed as transmitters, sending an example data packet similar to the data that was eventually used once deployed. One of the transmitters used a simple PCB antenna (Figure 6) while the other used a higher range whip style antenna (Figure 7). Then I programmed two challengers as receivers, again one had 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 to 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) [20].

Nine different distances from transmitter to receiver were tested, in 200m increments starting from 0m as a baseline to a distance of 1600m (Figure 13). An important aspect in getting a successful LoRa connection is whether there is line of sight between the transmitter and receiver. Figure 14 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. However the effects of reflections are virtually impossible to account for in the real world and therefore no accurate predictions could be made.

Figure 13: Google earth image of data collection points

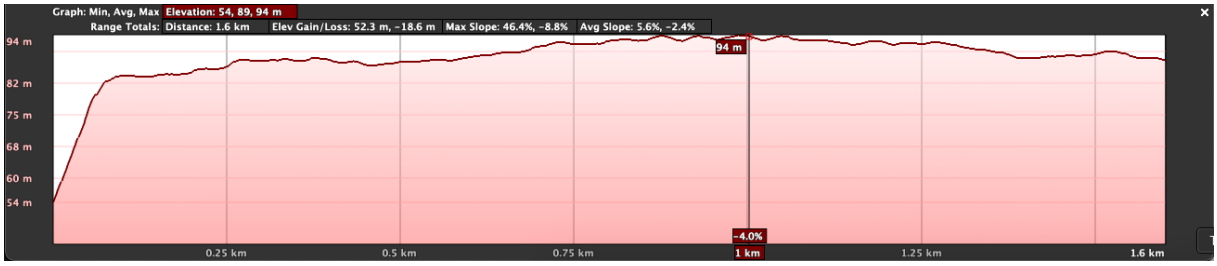


Figure 14: Elevation profile of test area (ignore large dip at start)

At the time of the actual test however 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

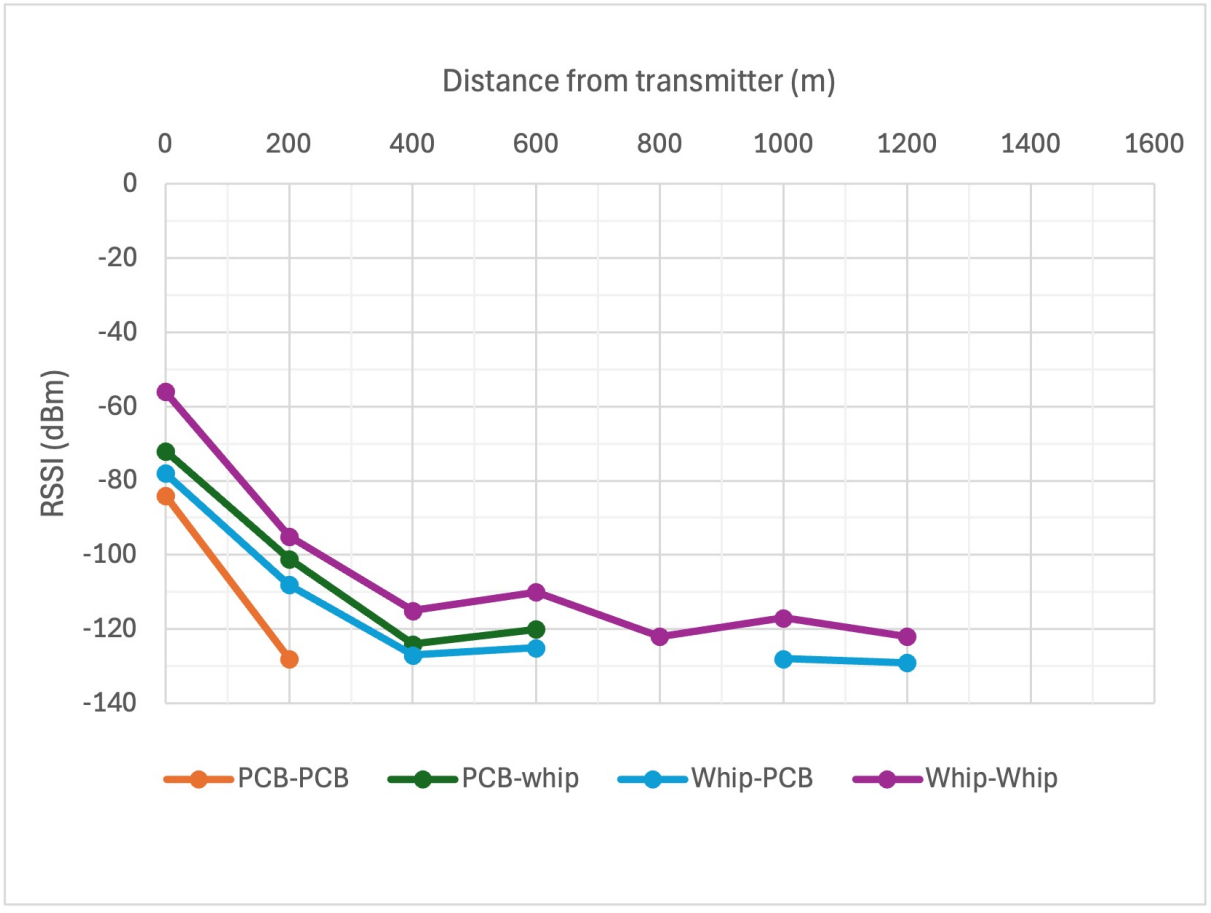


Figure 15: Graph to show signal loss from different receiver and transmitter configurations (higher is better)

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 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.

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 particularly useful 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 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.

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 day to able to run continuously. It would be tempting to then say that the node would need less than 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 it's 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, leaving a total sunlight requirement each day of 4h:03m.

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:

- 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 30-40 degrees to optimise solar efficiency.
- 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 needs. 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 the to allow for external component wires. 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 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 [21]. This level of protection should be more than adequate to withstand the heavy rainfall it will experience.

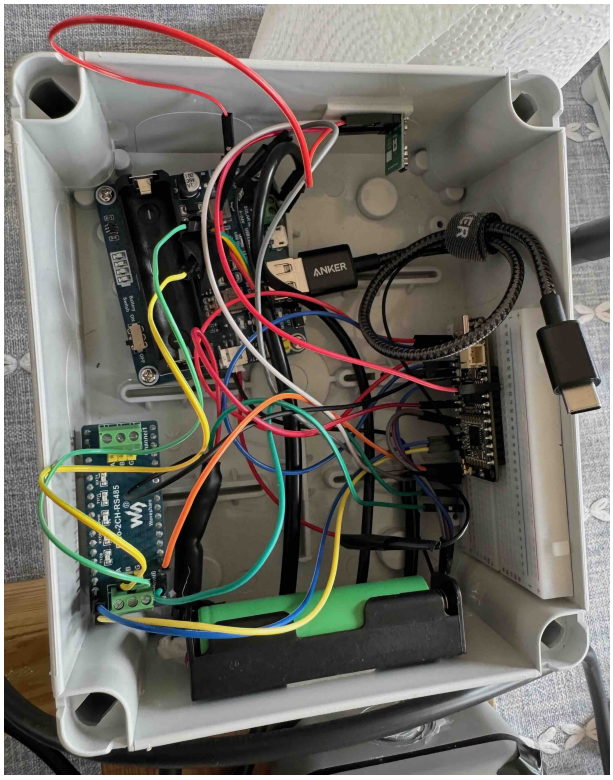


Figure 16: Open junction box showing internal wiring

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 then 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 then 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 as to 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 collecting its readings was secured 2m above it. 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 17, allowing for the sensor to be inserted into the ground.

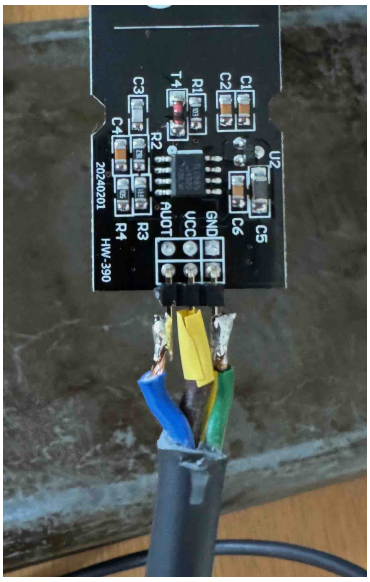


Figure 17: Soldered soil moisture sensor

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 9). This means it was essential to waterproof this part of the board.

To achieve this I followed an online tutorial on a hobbyist website [22]. 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 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. 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 south 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 positioned at a height of 2m.

ADD final assembled box on a pole with annotations

7.4.4 Gateway

The gateway node 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 persons private home, the aim was therefore to make a minimally imposing and so a design similar to an internet router was chosen.

The two devices inside the box would be the raspberry pi used to upload data and the challenger which received incoming LoRa packets from the repeater. 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 any 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 my garden and ran continuously from the 15th of August.

8.1 Challenges and solutions from test deployment

One of the first challenges from the test deployment was the fact the batteries on the nodes were not quite sufficient to allow 24/7 readings. On the fourth day of the run, after an overcast evening the night before, one of the nodes stopped transmitting at around 6am - just before sunrise. To remedy this another battery was fitted to each of the end nodes (but not the repeater as this had much lower battery consumption from the lack of any sensors). This doubling of battery capacity made loss of power during the night much less likely. The fact the solar panels could charge the battery fully by mid-morning pointed to the fact that solar capacity was sufficient but battery capacity was not.

A related problem was that when the node died it did not start repeating again after

this despite solar power returning and the battery getting recharged. I discovered that when the microcontroller detected brownout - being 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 without modification would essentially run infinitely 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 [23]. This version of the safemode.py file was made specifically for the case I was using i.e. remote solar powered projects where 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 a state where a hard manual reset was necessary. This problem is discussed further in the evaluation.

A seperate 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 was made to surround the sensor. Consisting a wooden frame and aluminium shield to block light falling on the junction box. While this method did help to reduce the delta at midday to around 3 celsius there was still clearly quite a pronounced effect. So to further reduce any warming the solar panel was mounted to the rear of the node. This would allow the temperature sensor to face north instead of south which in the northern hemisphere would result in much reduced solar warming to the sensors box as the sun tracks over the southern area of the sky in the UK.

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 circuit python 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 challenger as well as example code.

9.0.1 The nodes

9.0.2 The repeater

9.0.3 The gateway

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.

ADD LoRa settings

LoRa has many parameters that must be aligned for two devices to communicate successfully. These include:

1. **Spreading factor:**
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. **Coding rate:**
5. **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. For the 868mhz band, the maximum effective radiated power that can be

used is 25mW. This corresponds to a transmit power of roughly 14dB.

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 12 a message may take approximately 1 second to send. 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 messages could be sent on a given day - or roughly 1 message every 2 minutes.

9.2 Sensor nodes

9.3 Repeater

9.4 Gateway

9.5 Remote diagnostics and error handling

Since all 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 technically network connected with LoRa there is no easy way to send firmware updates over the air without developing a program to accept and create new files over LoRa (FACT CHECK THAT), which was outside the scope of a three month dissertation. Therefore instead the nodes would need to have robust error handling written into their programs.

Diagnostics on the gateway node was luckily much simpler as this was powered on 24/7 and network connected. (EXPLANATION OF TAILWIND AND RVC VIEWER)

10 Overview of software design

11 Hosting

Both the backend and frontend were hosted on Render. 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 for a 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

reduce the cost to £10.

12 Backend

12.1 postgreSQL database

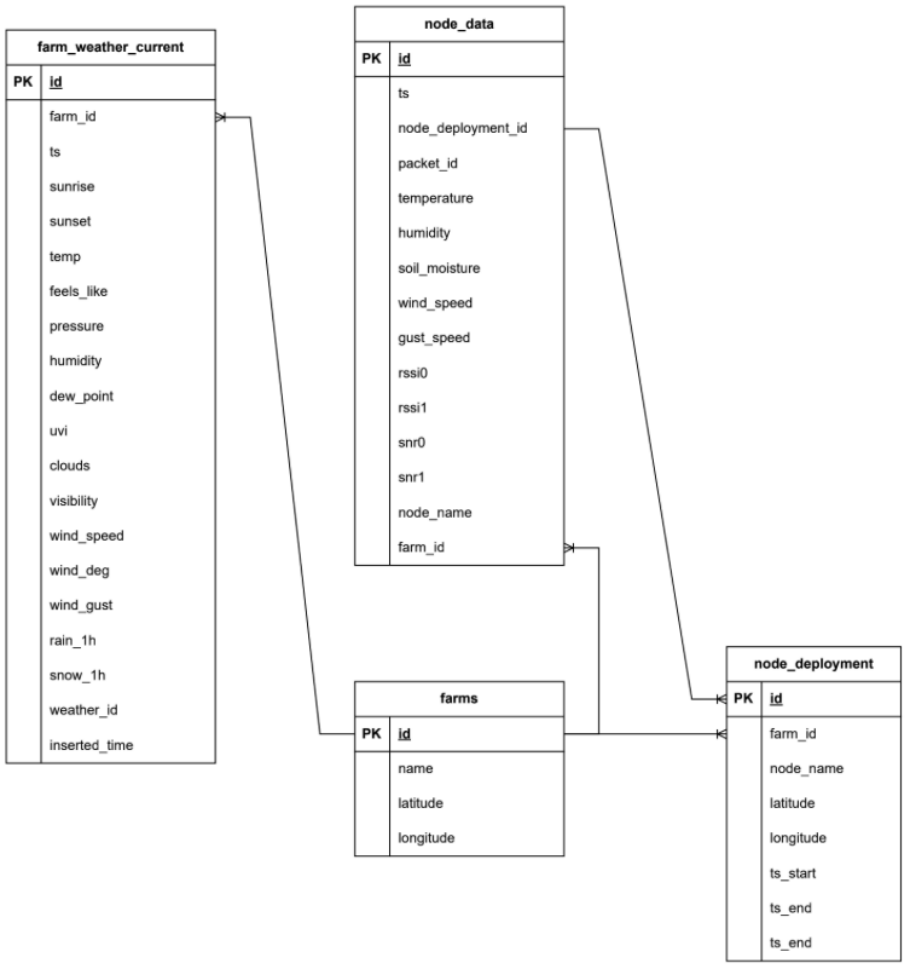


Figure 18: Table schema for postgreSQL database

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 below) 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 strictly 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, Node.js, and express. 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 above.

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 on. 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. JavaScript) 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: 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 packet 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 gives useful 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 to pool requests together on the same connection. This helped to improve the speed of SQL lookups and inserts helping requests to be executed faster, which in turn improved the perceived performance of the webapp.

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.

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 served to the browser and are injected separately at deploy time. This prevents the information from leaking into the public domain.

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) [24]. 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.

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 table_names like so (psuedo-code for clarity);

```
const white_listed_tables = new Set('node_data');

if (!white_listed_tables.has(httpBody.table_name)) {
    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. 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 to set up a 10 minute job on my host server. This then called related functions to get both current and forecasted data for use with my machine learning model.

13 Frontend webapp

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

13.1 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.

13.2 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.

13.3 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 over a datapoint and see the tool tip appear where they are hovering, on mobile the digit used to select may cover important tooltip information. By making sure the tooltip hovers slightly away from the selection this is no longer a problem.
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. Therefore I developed a number of functions to improve readability on mobile devices by dynamically decreasing chart height and font size for mobile screen sizes.
3.

13.4 Walkthrough of features

Walkthrough of main features with images and

14 Forecasting with machine learning

With the webapp and database operational I moved on to attempting to forecast the microclimate with machine learning. The idea for this model would be that I could give it future general weather forecast data and then it would return adjusted microclimate datapoints based on this.

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 [25].

Both LightGBM and XGBoost are known as 'Gradient Boosted Machines'. This is a relatively new 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 [26]. 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 algorthim for my use case.

To perform the training I installed python with the lightGBM and m2cgen libraries as described above. I then also installed pandas for data handling purposes.

14.2 Machine learning process

I aimed to create eight separate machine learning models, one for each node (node 1 and node 2) and one for each sensor type per node (temperature, humidity, wind speed and soil moisture). Training the model would require "features" which are essentially the inputs to the model that it then uses to estimate a "target" (i.e. the output).

The final models would take a 48 hour general future forecast as their input and then output a predicted microclimate specific sensor reading (e.g. predicted node 1 temperature). The models would be trained using historic current weather collected from the same api and trained against actual sensor readings to create an accurate prediction. The following diagram shows a process map this training process:

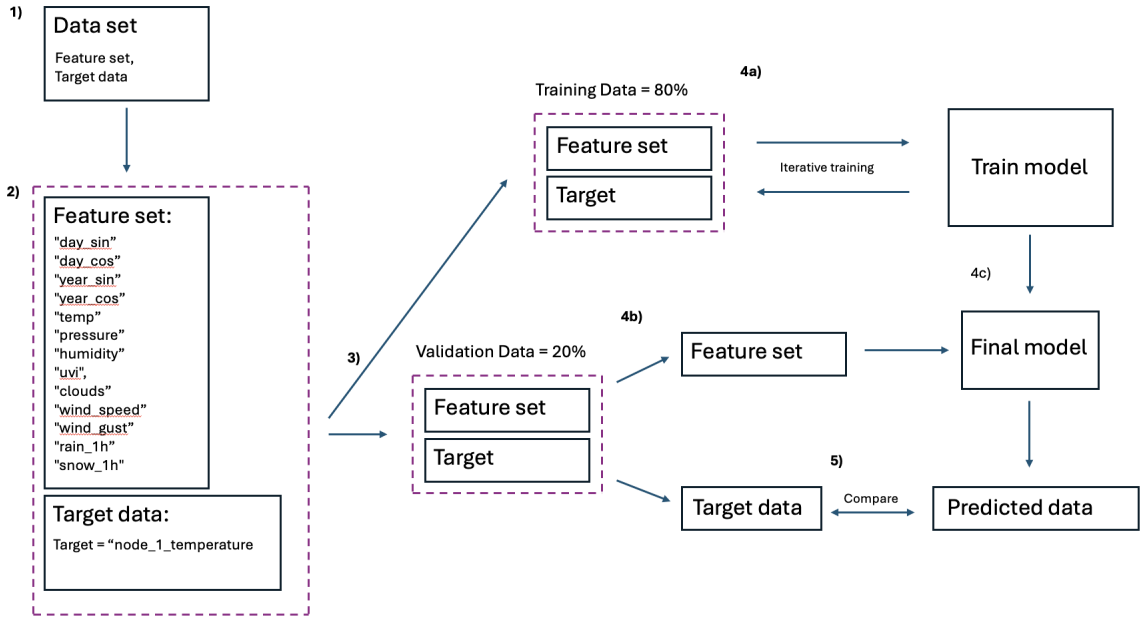


Figure 19: Infographic showing steps for training with LightGBM. This uses the temperature target for illustration.

1. Preparing the dataset: A single cleaned dataset was created with timestamps between the api weather data and node data matched. As API readings are taken every 10 minutes and node readings every 1 minute, 9/10 node readings were discarded. The final dataset was roughly 1,400 rows.
2. Define the feature set and target data: The feature set (weather API) and target (node data) were defined, and unnecessary columns discarded. The Unix timestamp used for database purposes was transformed into sine and cosine representations of day and year. This is necessary when training 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.
3. Split the dataset into training (80%) and validation (20%): This step is necessary so that s
4. (a) Run iterative training model: A model is trained against the feature set. It uses the target as supervision to get the best possible model.

   (b) Remove target data from feature set: The validation data is split into feature set and target. This is so the final model can be tested without access to the actual data.

   (c) Produce a trained model: When the model training has iterated at least 50 times with no improvements a final model is generated.
5. Use validation feature set in final model and compare predicted data with actual: This step then compares how the model performs on the validation data compared to its performance during training. In most instances the model will be less accurate on validation data as it has never seen this before.

With the eight models created they were then placed into the backend of the webapp and fed new forecast data every 10 minutes to create regularly updated predictions. The front end requests the predicted data from the backend and displays this as a chart line.

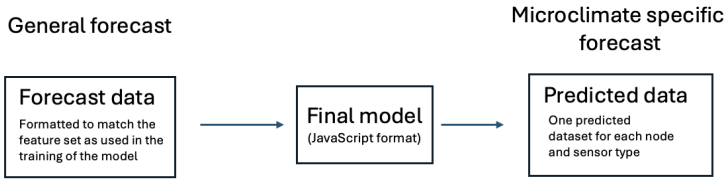


Figure 20: Infographic showing how final model is used on the webapp

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 significantly 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 placement, 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 $\approx$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. For example 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 [27]. Consequently, from the 15th of August to the 28th 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 28th August to the 2nd September Chipping Sodbury - where the nodes are currently placed - received roughly 3.4cm of rain according to weather forecasts. 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 challengers and antennas used in this configuration are able to reach at least 1200m of range in non-ideal circumstances.

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 power of the sensor nodes is cut off at night then it is not guaranteed that the node will restart the code.py program in the morning. 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 is ultimately appears to be a firmware issue there is not an obvious or easy solution for this.

With enough time I would probably rewrite all of the challenger software in Micropython and see if that firmware behaves in a more useful way on power outage. Unfortunately the issue was picked up fairly late in development and it was unrealistic to perform a refactor of this scale with the time left.

15.2 Quantitative comparison with commercial alternatives

Section will compare my node to others in a table and conclude on strengths and weaknesses of each

16 Web app: System Usability Survey

Section will show results from SUS survey and conclude on strengths and what could be improved about the web app in general. Also user stories for future action that came out of the survey.

16.1 Methodology

17 Accuracy of machine learning model

Section will compare machine learning models prediction to the actual sensor data and the general forecast and conclude on whether model is appropriate for application

17.1 Methodology

18 Future work

Future improvements for weather station and app

19 Closing remarks

Brief conclusion

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] The Things Network Global Team, *New LoRa world record: 1336 km / 830 mi*, https://www.thethingsnetwork.org/article/new-lora-world-record-1336-km-830-mi, Sep. 2023.

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

[4] 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.

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

[6] 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.

[7] S. Sinha, *State of iot 2024: Number of connected iot devices growing 13% to 18.8 billion globally*, https://iot-analytics.com/number-connected-iot-devices/, Sep. 2024.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

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

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

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

[24] 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.

[25] S. Saha. "Xgboost vs lightgbm: How are they different." Accessed: 2025-08-27.

[26] B. Tuychiev. "A guide to the gradient boosting algorithm." Accessed: 2025-08-27.

[27] 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.

Part V

Appendices

A Breakdown of costs

Category	Name	Units	Cost (£)	Total cost (£)
Electronics	iLabs Challenger RP2040 LoRa (868MHz)	4	20.80	83.20
Electronics	Solar Power Management Module for 6V-24V Solar Panel	3	9.70	29.10
Electronics	RS485 Wind Speed Transmitter	2	43.20	86.40
Electronics	Capacitive Soil Moisture Sensor	2	4.00	8.00
Electronics	Monocrystalline Silicon Solar Panel	3	9.90	29.70
Electronics	2-Channel RS485 Module for Raspberry Pi Pico	2	7.50	15.00
Electronics	9V Step-Up Voltage Regulator U3V40F9	2	9.60	19.20
Electronics	18650 Lithium-ion Rechargeable Cell - 2500mAh 3.7V 30A	5	3.99	19.95
Electronics	Raspberry Pi	1	33.60	33.60
Electronics	iLabs LoRa Antenna (EU868)	4	5.53	22.12
Electronics	DHT11 Temperature-Humidity Sensor	2	3.90	7.80
Electronics	Half size breadboards	3	2.00	6.00
Electronics	5 Meters of Black 3 core cable	1	5.00	5.00
Electronics	Battery storage case	5	1.00	5.00
Electronics	Assorted (M-M, M-F, F-F) jumper cables	1	6.00	6.00
Enclosure	IP65 junction box	3	12.18	36.54
Enclosure	Small IP55 junction box	2	4.20	8.40
Enclosure	Wooden posts (2m)	3	17.00	51.00
Enclosure	Gravel board	1	8.50	8.50
Enclosure	Steel spur post supports	3	8.95	26.85
				507.36

Figure 21: Table of component costs

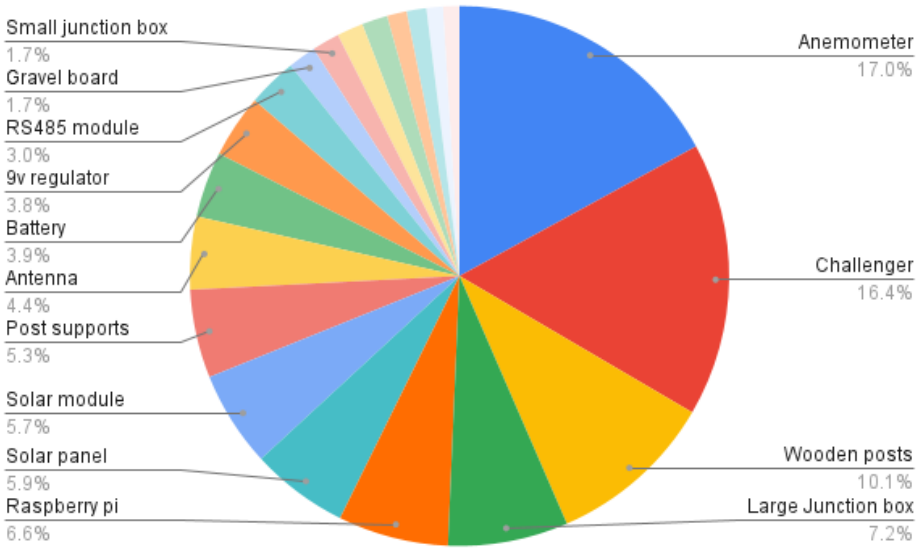


Figure 22: Chart to visualise relative cost of components

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 (9 May 2025)