Time Series Analysis for Environmental Monitoring and Forecasting Using TinyML

Presented by:
Takura Tyrone Kawome

Prepared for:
Dr. Joyce Mwangama
Dept. of Electrical and Electronics Engineering
University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Bachelor of Science degree in Electrical and Computer Engineering

April 14, 2023

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signature:.....
T. T. Kawome

Date:.....

Acknowledgments

- First I would like to thank my wonderful supervisor, Dr Joyce Mwangama for the guidance and support and being readily available to assist.
- I would also like to thank my parents for the love and support they continue to give me.
- My aunts Tariro and Chipo Mazhetese for making all this possible
- Tavonga Mazhetese for some of the components used for testing my system
- My friends Dean Makoni and Kudzai Samakange for the wonderful journey these past four years

Abstract

TinyML is a technology that allows machine learning models to run on low-power devices such as microcontrollers. The model is run locally on the device, eliminating the need to run it on the cloud. This research aimed to investigate the use of TinyML in Environmental Monitoring and forecasting and to produce a working prototype for this system. The use case selected was temperature monitoring and forecasting to automate regulating greenhouse environmental conditions.

The machine learning model was trained on the cloud, Google Colab, were there are high computational resources, using Python's TensorFlow library. The Convolutional Neural Network (CNN), Multilayer Perceptron (MLP), and the Long-Short Term Memory (LSTM) were used to train the model and were tested for prediction accuracy. Metrics of accuracy used were the Root Mean Squared Error (RMSE) and the Mean Absolute Error (MAE). The trained models were converted to their lightweight versions, TensorFlow Lite, to be deployed on the target microcontroller for inference.

The developed system could predict temperature for a single time step into the future using historical temperature of the previous five time steps. Predictions were accurate with an RMSE value of 1.45 and an MAE value of 1.3.

Contents

List of Figures

List of Tables

Terms of Reference

AI - Artificial Intelligence
BLE - Bluetooth Low Energy
CNN - Convolutional Neural Network
CPU - Central Processing Unit
DNN - Deep Neural Network
FPU - Floating-Point Unit
IDE - Integrated Development Environment
IoT - Internet of Things
LSTM - Long-Short Term Memory
MAE - Mean Absolute Error
ML - Machine Learning
MLP - Mulitilayer Perceptron
NN - Neural Network
RMSE - Root Mean Squared Error
RNN - Recurrent Neural Network
(S)ARIMA - (Seasonal) Autoregressive Integrated Moving Average
SVM - Support Vector Machines
TinyML - Tiny Machine Learning
TSA - Time Series Analysis
TSF - Time Series Forecasting

Key Words

TensorFlow (Lite), Inference, Edge Computing, Cloud Computing, Real-Time Machine Learning, Time Series, Environmental Monitoring and Forecasting

Chapter 1

Introduction

1.1 Background to the study

Tiny Machine Learning (TinyML) is a relatively new and fast-growing technique aimed at low-power systems such as Raspberry Pi/Arduino platforms in the embedded systems environment [1]. Machine Learning models are optimised to run on low-power microcontrollers. In recent years there has been an increase in the use of small and smart IoT devices. With the rise of machine learning, there has come the need to merge these two, that is, running machine learning algorithms on IoT devices for different applications. TinyML offers that intersection of Machine Learning and the Internet of Things.

Machine Learning algorithms are normally executed on the cloud, and the results are then sent to the desired location. This brings about challenges such as communication latency due to the transfer of data. With this continuous transfer of data, the Internet becomes saturated. The security of the data is also at risk due to it being public, making it prone to attacks or interceptions by hackers. Running the Machine Learning algorithms on edge devices solves the communication and privacy issues as the data is local to the device and secure with no need to transfer it. Local processing eliminates the need to send data through the network, which reduces data costs. Latency is reduced as it takes time to transmit data and wait for decisions.

There is a wide and growing spectrum of Machine Learning applications that are fuelling the advancement of TinyML. In healthcare, small wearable devices such as smartwatches have many sensors embedded in them to monitor vitals. Video analytics in surveillance applications use camera sensors for fast detection. Smart IoT devices such as digital

assistants, industrial monitoring and control and smart energy grids and intelligent transportation systems all require always-on-devices for continuous operation. Low-power devices require a single charge to operate for longer periods of time. TinyML is optimised to operate on these devices with power consumption in the milliwatts [2].

Most Machine Learning applications, such as those mentioned above, require real-time operation hence execution should be fast. This brings about the concept of edge computing where all the processing is done at the network edge by the devices (nodes). Computation at the network edge gives advantages in speed, cost and power efficiency, privacy of data and allows autonomy of the applications being executed. The goal is to democratise Artificial Intelligence where anyone can get access to cheap hardware. There won't be the need of acquiring cloud services which are expensive but rather using cheap, simple and accessible development tools to develop Machine Learning Models.

Most Artificial Intelligence (AI) applications in Africa are cloud-based. People do not get the benefit of these AI applications because of the lack of internet connectivity and its costs, and high energy requirements of the computational resources [1]. TinyML aids in the Sustainable Development Goals set by the United Nations [3] for industry growth and innovation. TinyML does not require a network infrastructure [1], making it easy to deploy for research environmental monitoring systems. With the advancement of Agriculture 4.0, which aims to improve product growth in the sector with the aid of precision farming using environmental monitoring systems [4], TinyML is a cheap solution for local farmers in rural areas or with greenhouse who cannot afford internet connection or have an unreliable connection. Not only will this environmental monitoring system help automate greenhouse farming or farming in general with minimal human supervision, but it will also collect important environmental data that can be used to optimise yield growth.

1.2 Objectives of this study

1.2.1 Problems to be investigated

The main objective of this study is to develop a Machine Learning model that uses real-time sensor data of the environment, such as temperature, for monitoring and forecasting of that environmental data. This model should be adapted into a lightweight version for execution on a low-power microcontroller. The objective is motivated by the growth in

use of small low-powered IoT devices. Cost of these devices is low, and they are easy to acquire. Adapting a Machine Learning model for a low powered device enables this application to be run efficiently with an always-on-device for continuous monitoring of environmental conditions. Running a trained Machine Learning model locally on a device reduces dependence on the cloud and eliminates latency issues when transmitting data through a congested network. The target application is using this system in a greenhouse where the predicted temperature is used to regulate the environmental conditions in advance for optimal conditions for plants/crops grown in the greenhouse.

1.2.2 Purpose of the study

The purpose of this study is to investigate the intersection of Machine Learning and edge IoT for the analysis of IoT related data. The Machine Learning model should be trained and adapted for execution on a microcontroller, Raspberry Pi/Arduino. The embedded platform prototype should be able to use real-time environment data, such as temperature, and make predictions which can be used to control temperature in greenhouses for instance.

Embedded platforms are constrained devices in terms of memory and power. Investigation of whether these constraints have any effect on the accuracy of the predictions will be required to help understand the trade-offs. Environmental monitoring and forecasting can be essential in developing an automated greenhouse system that can control the environmental conditions to suit whatever crop is being grown. Accuracy of the predictions in this use case is very vital for the healthy growth of the plant/crops. Inaccurate predictions can provide unfavourable conditions for the plants which could kill them.

1.2.3 Project Deliverables

The user requirements of this project involve producing a working prototype of the Machine Learning model being adapted for execution on a power efficient microcontroller.

1. Train the model in the cloud where high-power computational resources are available.
2. Adapt the model to a lightweight environment. For instance, migrate from TensorFlow to TensorFlow Lite.

3. Run the inference on a particular microcontroller such as through the Arduino library.

1.2.4 Proposed Research Objectives

Based on the above user requirements, the objectives of this project are to test two Graduate Attributes (GA's) which are problem solving, this includes the ability to identify and define the problem statement and coming up with a viable solution to solve it. The second GA includes investigations, experiments, and analysis. This involves testing the possible solutions and evaluating their performance to select and justify the best solution.

1. Analyse requirements for embedded system design.
2. To design and implement monitoring and forecasting which optimizes resource performance given huge restrictions in available networking and computing performance of hardware
3. An investigation into the tools required to design and implement a forecasting and monitoring system
4. Analyse and optimize the system to function on an embedded device
5. Investigate the trade-offs of accurate predictions vs constraint device resources

1.3 Scope and Limitations

The scope of this project is to focus on the performance of the microcontroller running the Machine Learning inference and testing if it gives accurate predictions. The model should be optimised to function on the embedded device.

One of the limitations is a time constraint. Since it is just a 12-week project, there will not be enough time to extensively train the model to continuously improve the forecasting of the environmental condition of interest. There are any environmental conditions that can be predicted such as temperature, pressure, and humidity but temperature prediction will be the only focus.

TensorFlow Lite, the tool that will be used to optimise the Machine Learning model

to run on a microcontroller only supports a few microcontrollers and of those, only one is available. Recent literature indicates the Arduino Nano 33 BLE Sense to be the best choice.

Due to budget constraints, another microcontroller that could be used to test the performance against could not be acquired. This limits the project to implementation on only one microcontroller.

Since the project will be dealing with time series data, this limits the choice of algorithms that could be used to train the model. Literature suggest ARIMA, SARIMA and LSTM algorithms to be suitable for time series data.

1.4 Plan of development

The plan of development will describe the structure of the project and give a roadmap of how the research will be executed. This is an overview of how the objectives are going to be achieved.

Chapter 2 will present a literature review that will give more context on TinyML and how it is used in time series analysis. This will highlight how other researchers have gone about different approaches. These approaches will be analysed and critiqued. A discussion on other tinyML applications will be discussed as well as the challenges and limitations of tinyML.

The plan and methodology that will be used to execute the solution will be presented in Chapter 3. This will include the steps taken to design the solution and discussion of the design considerations. Acceptance tests will be given to define the success of implementation of the solution.

Chapter 4 will present the results of implementation of the different algorithms and tests that will be highlighted in Chapter 3.

Chapter 5 will provide a discussion of results presented in the previous chapter. An analysis of these results will be done and an explanation of what they mean. The best algorithm that performs the best will be evaluated.

Chapter 6 will give a conclusion on the scope and finding of the project.

Chapter 7 will give recommendations for future development of the topic.

Chapter 2

Literature Review

This literature review aims to investigate and better understand the implementation of Tiny Machine Learning. It will evaluate available literature based on its various applications and pay particular attention to time series analysis.

This chapter will provide a brief overview of Tiny Machine Learning and Time Series Analysis. It will proceed to highlight the opportunities and challenges of implementing Tiny Machine Learning and discuss its various applications. An in-depth analysis of the system setup and tools required will be done. This review will attempt to achieve a low-level description of the topic for readers who need a background in machine learning.

2.1 Overview of IoT and Tiny Machine Learning

2.1.1 Internet of Thongs and Cloud Computing

The Internet of Things, (IoT), refers to a large network of devices connected to the Internet which are collecting and sharing data. Ubiquitous Internet connectivity and advancements in technology have enabled cheaper production of computer chips which has increased the number of IoT devices [5]. Figure 2.1 shows the number of connected devices where in the year 2019 8.6 billion devices were connected. The number of connected devices is increasing exponentially by the year with a projected number of 29.4 billion connected devices by the year 2030 [6]. These devices have been integrated to be part of the daily lives of end-users. An example are wearable devices such as smart

watches and sensor devices capable of environmental monitoring providing information on temperature, humidity, atmospheric pressure, air quality and more.

With the aid of cloud computing, IoT devices are able to communicate with each other anywhere in the world as long as they have Internet connectivity. Cloud computing services also enable real-time computations and communication of the information to all devices that are granted access. However, cloud computing has limitations in that devices need to have Internet connectivity and there are latency issues when a device is communicating with a remoter server or data centre.

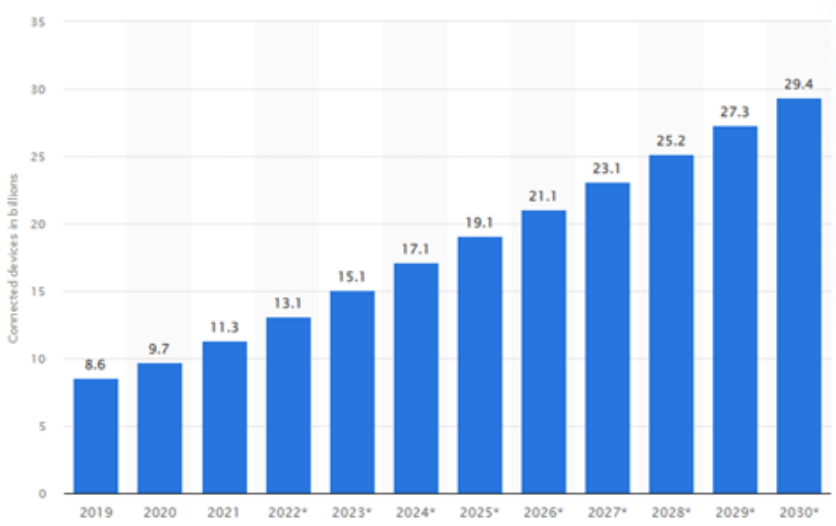

Figure 2.1: Projection of Connected IoT Devices from 2019 to 2030 [6]

2.1.2 Edge Computing and Real Time Operations

Edge computing aims to provide better response time and improved availability of bandwidth for devices. Edge computing is a distributed computing framework that brings computation closer to data sources, that is, the edge (IoT) devices or local servers [7]. With the large number of IoT devices, large amounts of data are being produced, processed, and communicated between different devices and servers. The development of edge computing provides an improved speed of real-time processing data obtained from device sensors. This is vital for some real-time systems where an almost instantaneous response is fundamental, a typical example being autonomous vehicles, allowing the system to make possible life or death decisions quickly.

Figure 2.2 below shows the basic architecture of edge computing with three essential layers, the cloud layer, edge layer and device layer. The cloud layer is the central

data centre that stores and processes computer resources. Access to this data centre requires Internet connectivity. Following the cloud layer is the edge layer. The edge layer decentralises the cloud layer with many distributed data centres with the same functions as the cloud server but at a smaller scale. The edge layer reduces the congestion of devices on the Internet and increases bandwidth availability. This is achieved by caching resources on the edge layer from the central cloud layer, bringing these resources closer to devices.

The final layer of the architecture is the device layer. The device layer involves sensors and controllers that are data sources that need to be processed and possibly communicated between different devices. Bringing processing closer to these devices reduces latency enabling quick response of systems based on the data received. This architecture is robust because devices can operate offline without depending on Internet connectivity to the cloud.

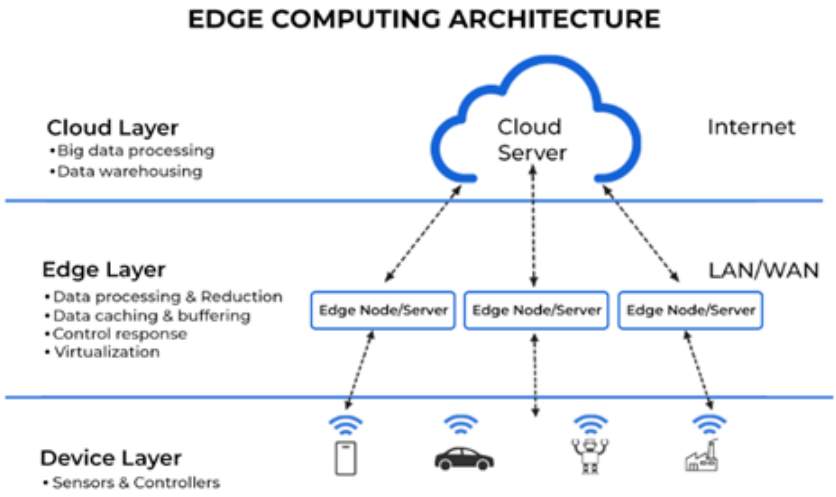

Figure 2.2: Edge Computing Architecture [8]

The application of edge computing with embedded devices is growing and becoming vital. Microcontrollers have integrated chips with central processing units (CPU), memory and interfaces that allow communication with sensors and controllers. Based on the application, the processor reads data from the sensor, which is sent to the communications module. The processor is also powerful enough to perform some processing tasks within the edge device, allowing the execution of real-time operations.

The real-time processing of data locally on the device is driven by the low and power-efficiency of embedded devices. Ultra-low-power microcontrollers enable the remote deployment of these devices so that they can operate without human supervision. Ultra-low-power microcontrollers use low sleep current where the system spends most of the

time in standby mode. The system wakes up periodically or asynchronously to perform specific tasks. Microcontrollers with 32-bit CPUs, moving from the 8-bit and 16-bit, are being manufactured, which achieve faster execution times, thereby spending less time processing tasks and saving more power.

2.1.3 Tiny Machine Learning (TinyML)

Tiny Machine Learning is the merging of the Internet of Things and Machine Learning technologies, utilising machine learning algorithms and IoT devices' hardware and software capabilities to analyse sensor data locally on the devices at extremely low power [9]. TinyML's fundamental goal is to use cross-layer design approaches that allow for adapting machine learning models for ultra-low-power devices such as microcontrollers that utilise milliwatts of power and can operate for a long time on a single charge. The three main advantages of TinyML are low latency, low energy consumption, and low bandwidth [10].

The usual approach for executing a machine learning model is to do it in the cloud, with significant computational resources and transmit the results to the desired destination. Using this approach brings the emergence of communication latency through the sending of data to different locations. Execution in TinyML does not require Internet, cloud, or edge connectivity because execution occurs on the target device.

TinyML was adopted for small IoT devices that use less power than standard Central Processing Units. An experiment was conducted to observe the power consumption of different hardware [11]. Figure 2.3 shows the results of the investigation. Even in the idle state, the CPU and GPU consume large amounts of power relative to the milliwatts consumed by the CPU of embedded devices.

Hardware	Idle (W)	Running (W)	Dynamic (W)	Inf/Sec	Joules/Inf
GPU	14.97	37.83	22.86	770.39	0.0298
CPU	17.01	28.48	11.47	1813.63	0.0063
Jetson	2.64	4.98	2.34	419	0.0056
Movidius	0.210	0.647	0.437	300	0.0015
Loihi	0.029	0.110	0.081	296	0.00027

Figure 2.3: Power Consumption and Energy Cost per Inference of different Hardware [11]

Building a TinyML project involves three main steps. The machine learning model should be trained on the cloud where high computational power is available. Secondly, the model should be modelled and optimised to run in a lightweight environment, that is, the chosen

microcontroller. The last step is running the inference on the target device [10].

2.2 Time Series Analysis

2.2.1 Overview

Time series analysis is a method used to analyse data points in a sequence that are collected over different time intervals, that is, years, months, weeks, days, hours, minutes, or seconds [12]. When analysing the data, consistent time intervals are recorded. This enables an analyst to see how data variables change with time, giving more information on the characteristics of data. For a more consistent and reliable representation of the data's features, many data points are required. This also reduces the effects of noise that is present in the data. Many data points also validate trends and patterns of the data, ensuring that they are not just outliers and represent the data's seasonal variance.

The main application of time series analysis is forecasting future events such as temperature in weather, economic forecasts, and sales in a business. Trends in historical data are used as inputs that aid in making future predictions.

2.2.2 Components of Time Series Data

Time series data has four main components that show the characteristics of the data. These are the trend, seasonality, cyclical and the irregularity of the data. A model must be able to learn both the linear and nonlinear characteristics of these components to produce accurate predictions.

1. **Trend:** shows the movement of time series data in fixed intervals over a long- or short-term duration. This movement can show a general upward or downward trajectory. The trend can be deterministic where there is a cause for the trend or stochastic where the trend could be random and unexplainable.
2. **Seasonality:** shows the characteristic movement of data which is common to a particular period in time. An example is temperature fluctuations or rainfall received in different periods of the year. This is in fixed time intervals, or fixed

frequency, but over a short-term duration. The pattern is usually repeatable to the specific "season" [13].

3. **Cyclical:** this is when time series data exhibit rises and falls but these fluctuations do not occur at a fixed frequency. They differ from the seasonality component in that the frequency is not fixed and occurrence is not dependent on a certain period of time in a calendar.

4. **Irregularity:** these are random variations that occur during short periods. These random variations are not related to a particular model and are unpredictable [14].

2.2.3 2.2.3. Machine Learning Algorithms for Time Series Forecasting for Environmental Monitoring

Machine learning has been widely used in many environmental monitoring and forecasting applications. Different algorithms can be used based on the application of the system. Convolutional Neural Networks (CNN), which are suited for computer vision applications, have been used in supervised learning to monitor and predict deforestation in Amazon forests by identifying land patterns in the area [15]. The K-means algorithm has also been used for the same application, but this algorithm enables unsupervised learning [16]. Other work in Air Quality monitoring and prediction using Multilayer Perceptron and SVM regression algorithm [17].

Time series models are trained using supervised learning. With supervised learning, machine learning algorithms are trained using labelled datasets which are used to accurately classify or predict outcomes. The data is expected to be in the form of samples with inputs and outputs. Convolutional and LSTM models require the samples to be a three-dimensional structure, (batch size, time, features).

Time series forecasting was traditionally dominated by linear methods. Such is the (S)ARIMA, (Seasonal) Autoregressive Integrated Moving Average (MA). This model contains parameters for season and trend, including the linear combination of seasonal past values and forecast errors. The ARIMA, however has some limitations:

- It does not support corrupt or missing data
- It does not take into account complex distributions which are joint when assuming linear relationships

- Only supports univariate data, just focusing on a single characteristic of the data to make predictions
- Can only perform single-step forecasts, where it can only make a prediction of only the next time step into the future.

With these limitations, non-linear Deep Neural Networks (DNN) such as the Multi-Layer Perceptron (MLP), Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) are more suited for time series forecasting. They can map the complex nonlinear relationships, deal with missing data and data with more than one input variable [18].

Multilayer Perceptron

Multilayer Perceptron approximate mapping functions from their input variable to their output variables. With this capability, they have an added advantage over linear methods in:

- They are robust to noise. Learning and predictions can be done even when there are missing values.
- They are able to approximate arbitrary non-linear functions
- Multivariate inputs where more than one input feature can be presented, therefore supporting multivariate forecasting.
- Multi-step forecasts can be done where predictions of more than one time step can be made

The number of input and output variables must be fixed and for a given input pattern, an output must be produced. The temporal dependence must be specified upfront when designing the model, which is a limitation because the dependence is almost always unknown. A detailed analysis in fixed form must be done to get the dependence [18].

Convolutional Neural Networks

Convolutional Neural Networks were designed to effectively work with image data. CNNs have many effective applications in Computer Vision like object localisation, image classification,

image captioning and more. The neural network directly operates on the raw data, pixel values (for images). The model is trained to learn to extract features from the raw data that are of interest to a specific application. This is representation learning which is an approach that allows the discovery of representations needed for feature classification or detection from the raw data [19]. This ability to automatically get features from the raw data can be used to solve time series forecasting problems. The sequential data can be taken as a one-dimensional image. CNNs have the same advantages as the Multilayer Perceptron, with the added advantage of its representational learning.

Recurrent Neural Networks

The Long Short-Term Memory (LSTM) Recurrent Neural Network can handle the explicit sequential order and relationship between observations during learning the mapping function of the inputs and outputs. MLPs and CNNs do not have this ability. LSTM not only maps inputs to outputs alone, but it is also able to learn the mapping function for the inputs over time to an output [18]. In addition to the advantages of the MLP and CNN neural networks for time series forecasting, RNNs can automatically learn the temporal dependence of the input data.

2.3 TinyML Frameworks

Different tools and libraries have been developed to run machine learning models on constrained embedded devices and theses libraries are in continuous development to support more devices. There are three categories of implementing TinyML frameworks. The first and common approach is converting an already trained model and adapting the model according to the restrictions of the target microcontroller. The machine learning libraries used to train the models are TensorFlow, Scikit-Learn or PyTorch [10].

The second approach involves the integration of the machine learning models within the target embedded device. This gives the device the capability to locally train the model and do necessary analysis. Machine learning models are generated from the data extracted from sensors by the microcontroller. Accuracy of the models is improved and enables continuous unsupervised learning. The third approach uses a dedicated co-processor. The main computing units of the machine learning task is supported by the dedicated co-processor. Computational performance is aimed to be made efficient.

The second and third approaches are not so common due to their complexity and high cost of microcontrollers with processors capable of carrying out the mentioned specialised processing tasks.

2.3.1 TensorFlow Lite for Microcontrollers

TensorFlow Lite is an open-source library that was developed by Google. It adapts TensorFlow models to lightweight models that can be executed on mobile and embedded devices. TensorFlow Lite has two main elements with the tools used to adapt the model. The first is the converter that is used to port the trained TensorFlow model to tflite file. It optimises the code to be runnable on the small, constrained devices. The second element is the interpreter which is used to run the optimised code generated by the converter.

The tflite file can be converted to a C byte array when the model is intended to run on an Arduino. A 32-bit processor is required to run the model. Successful tests have been done on devices with the ARM Cortex-M series. Currently the only Arduino board supported is the Arduino Nano 33 BLE Sense. Another common architecture is the ESP32. More devices are beginning to get support to run Tensorflow Lite [20].

2.4 Applications of TinyML

TinyML has a wide application spectrum ranging from eHealth, industrial processes, smart things and spaces and environmental monitoring. These applications need the TinyML system to be efficient as they require real-time processing and execution of tasks. Initially TinyML applications were being developed for computer vision involving image recognition and classification which can also be applied in some security and surveillance applications. Later, development was widened to data analytics, language processing and predictive models.

Figure 2.4: Applications of TinyML

Most common applications in healthcare are wearable devices like smart watches which have sensors that monitor vitals like heart rate. It is important that real-time and accurate presentation of the vitals being monitored is produced. Due to the sensitivity of a user's medical information, privacy and security of the data is a priority. With TinyML operating locally on the device, privacy is ensured.

Automated greenhouse control systems have been developed using cheap and simple embedded devices. These systems help to minimise human interference to grow plants in a greenhouse. Environmental conditions are monitored and maintained to be favourable for the growth of the plants. With most systems, when temperature decreases the temperature produced by the heater is increased and when temperature increases, the shading system and ventilation conditions are adjusted accordingly. These changes are based on the actual current temperature being measured. Using TinyML with its predictive capabilities, the control system can adjust the greenhouse environmental conditions imminently, improving the system's response.

2.5 Opportunities

The implementation of TinyML brings improvements in the smart and embedded devices space. Artificial Intelligence is democratised as its implementation no longer requires using expensive hardware, design tools and cloud services [10]. This will foster adoption of artificial intelligence systems in rural and remote areas. As a result, giving life to the local economy of these communities.

An energy efficient system that can be powered by a battery and operate for a long time on a single charge is easy to deploy. With its independence from a power grid, deployment in less developed areas with less infrastructure is made possible. This allows the expansion of smart spaces.

TinyML increases reliability and the security of data. In applications that involve the use of big data, transmission of data from end devices to the cloud, where there are enough computation resources, is required. This wireless communication is through a lossy channel and consumes large amounts of bandwidth. Wireless transmission of information is prone to transmission errors and cyber-attacks in the form of eavesdropping or "man-in-the-middle" which compromises the data [10]. Doing all the processing on the device avoids encountering these issues as no transmission is required, making the data more reliable and secure from cyber-attacks. This also reduces latency and enable real-time execution of tasks.

2.6 Challenges

The TinyML software framework allows for easy integration of different machine learning algorithms and their optimisation in all the neural network's layers. TensorFlow developed a Lite and a Micro edition for IoT and embedded devices. Other examples are microTVM for embedded microcontrollers and TinyEngine which is based on OpenGL wrappers [9]. These frameworks make it easier to design machine learning models and optimise them without knowledge of hardware specifications. The main challenge is that not a lot of embedded microcontrollers support these frameworks. For instance, the Arduino Nano 33 BLE Sense is the only Arduino Board that supports TensorFlow Lite. Since TinyML is a fairly new approach, there is little documentation on how to implement its different applications.

Another challenge is device heterogeneity. The devices used to implement TinyML applications have different specifications from their processing power, power consumption, storage, memory, and different communication protocols. These devices require different setups and are no generic TinyML tools that can be used to facilitate that. This obstacle also complicates the development of a generic benchmarking methodology as it is difficult to determine the cross-layer metrics used to assess the performance of the TinyML application.

Data is very vital for machine learning applications. There is limited access to datasets that can be used to develop models. Machine learning model designers rely heavily on open-source datasets that are available publicly to develop their models. To improve the performance of a model, the model must be trained and optimised using external computational resources and then redeployed on the microcontroller. This is inefficient as training and optimisation cannot be done on an embedded device due to its computational constraints.

2.7 Related Work

Similar work was done [10], where TinyML was used to forecast temperature. The aim was to come up with a low latency, low bandwidth, and low energy solution to run a Machine learning model for a greenhouse application. The standard method of implementation was used, where the model was trained on the cloud where computational resources are high. After training, the model was adapted to a lighter version, that is converted from TensorFlow to TensorFlow Lite (tflite file). The tflite file was converted to a C byte array to make the model run on an Arduino device, the Arduino Nano 33 BLE Sense, using the Arduino IDE.

The target microcontrollers for the converted models were the Arduino Nano 33 and Raspberry Pi 4. Performance of the models on these two devices was compared. Two Neural Networks, the Multilayer Perceptron, and the Convolutional Neural Network, were used to train the models. The models were trained by two time series datasets of 15-minute and 1-hour intervals for each Neural Network. The datasets were collected from the greenhouse that was to be monitored. The first test was the execution time, that is the time it takes to process the data and make a prediction. The two Neural Networks, the MLP and CNN, each for the 15-minute and 1-hour datasets were deployed on the microcontrollers and the execution times were recorded. The results of the test are shown in Figure 2.5 and Figure 2.6 below.

As observed from both models, the Arduino takes more time to execute. This is due to its weaker processor relative to the Raspberry Pi 4. The Arduino has a 64 MHz CPU with 1 MB storage and 256 KB RAM. The Raspberry Pi 4 has a 1.5 GHz 64-bit quad-core processor with 1 MB RAM and an SD card slot that is used for storage of gigabytes of data.

Execution time for the MLP is faster than that of the CNN. MLP is a much simpler model with less operations than the CNN. The 1-hour datasets execute faster than the 15-minute datasets because the 1-hour datasets produce lighter models. This is because of the smaller number of datapoints that the 1-hour datasets have.

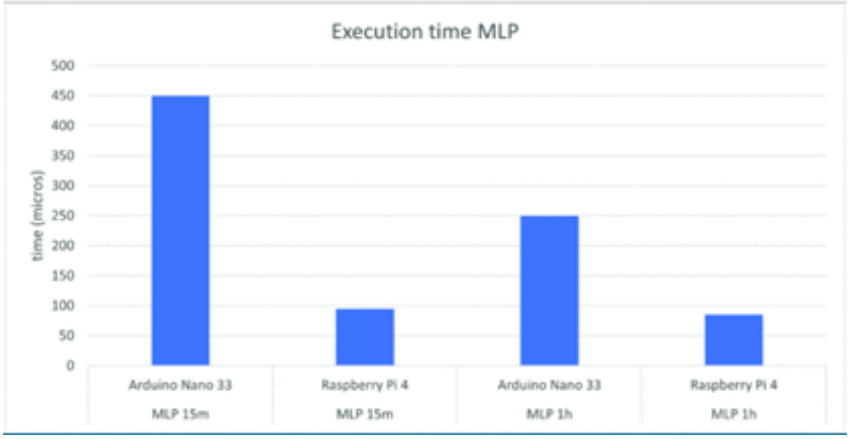

Figure 2.5: Execution Time of MLP models [10]

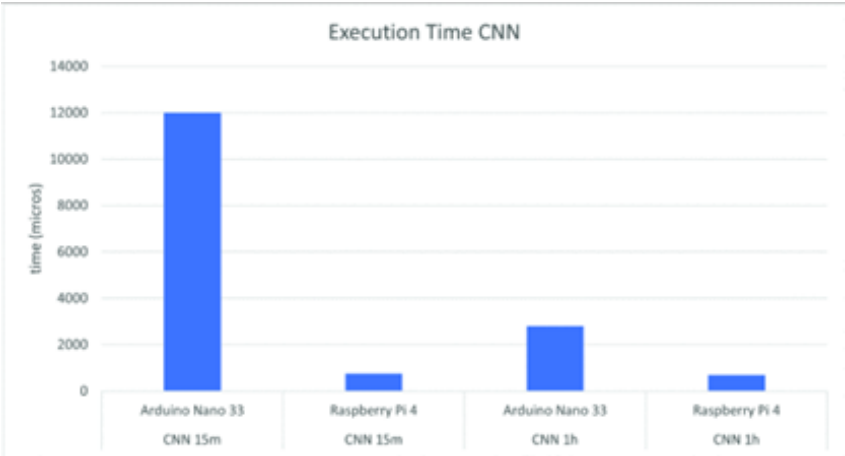

Figure 2.6: Execution Time of the CNN models [10]

The next test was the energy consumed by each microcontroller when running the inference. Energy consumed depends on the processor and the complexity of the model where a simpler model requires low processing power. Power consumption is also based on the hardware of the microcontrollers. Results of the tests are shown in Figure 2.7 and Figure

2.8 below. Although the Arduino has a longer execution time, it consumes less energy when running inference. These results were expected since the Raspberry Pi 4 has a stronger processor than the Arduino.

These tests only show hardware performance and not the accuracy of the predictions made by the models. This leads to the next test to show performance of the MLP, and CNN model based on the Root Mean Square Error and the Mean Absolute Error metrics.

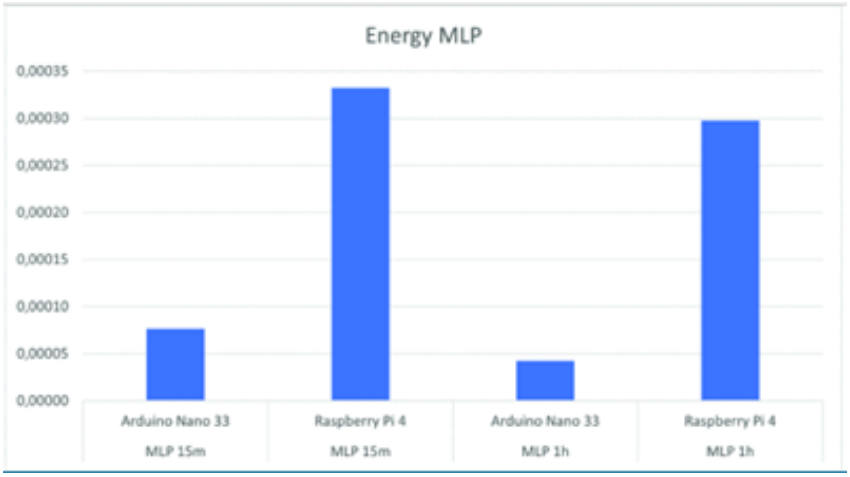

Figure 2.7: Energy of MLP models with Arduino Nano 33 and Raspberry Pi 4 [10]

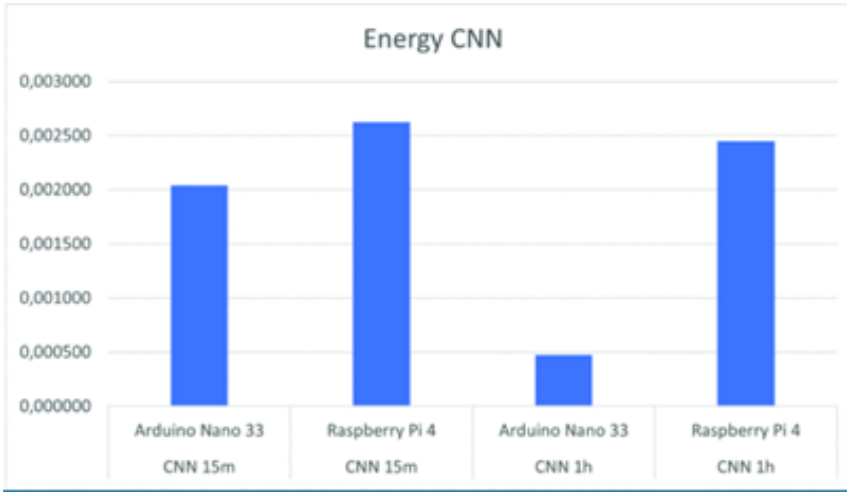

Figure 2.8: Energy of CNN models with Arduino Nano 33 and Raspberry Pi 4 [10]

The results for the quality of predictions made by the MLP and CNN models are shown in Figure 2.9 and Figure 2.10 below. The results are expected as the CNN model performs better than the MLP producing better predictions. Since the 15-minute models are trained with more datapoints, they produce more accurate predictions than the models trained by the 1-hour datasets.

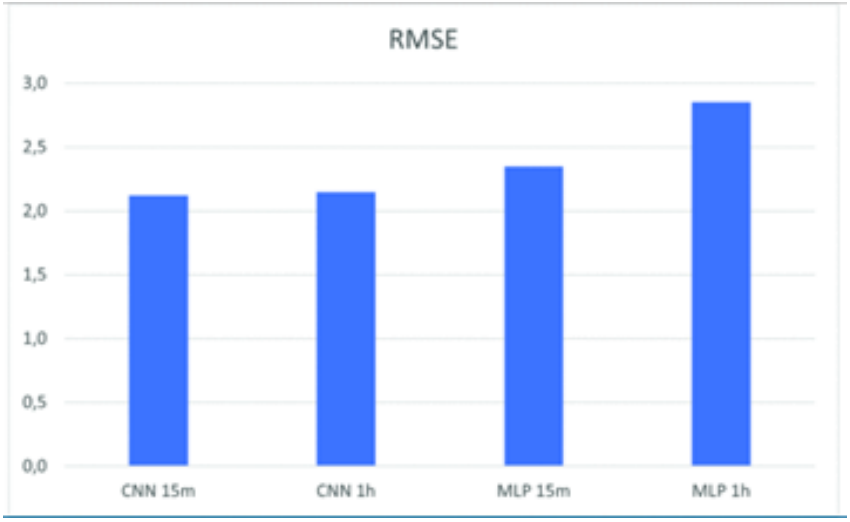

Figure 2.9: Root Mean Square Error [10]

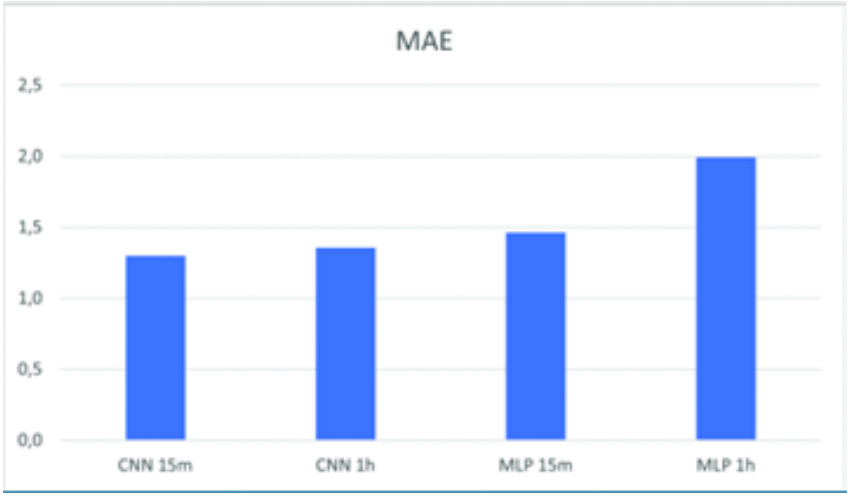

Figure 2.10: Mean Absolute Error [10]

The method of implementation does not show what the datasets used to train the models look like in terms of the timestamps and how they are analysed and refined to make sure the data is valid and clean enough to train the models. There was no mention of normalisation and how it affects stability and performance of the trained models by using transformation to put a model's features on the same scale. The system did not have a data collection subsystem that would allow for collecting the actual and predicted temperature readings to observe how the system is performing in the greenhouse. This data could be useful to train another better model which will be deployed on the microcontroller since the existing model on the microcontroller cannot be retrained.

Upon conversion to TensorFlow Lite, there were no tests to confirm the accuracy of the

lightweight model for comparison with the original model. The quantisation used, floating point or integer, was not highlighted and how each affect the accuracy of the TensorFlow Lite model.

Finally, only tests for the MLP and CNN Neural Networks were done and there was no mention of how a Recurrent Neural Network can be used in a TinyML application.

2.8 Future Roadmap

TinyML designers need to gather information on the data being used and how it is used and communicated across devices. This will enable the development of systems with the pre-installed software. Using this approach will eliminate the need of creating systems which depend on network requirements for their development. a system that is never networked is safe from cyber-attacks and ensures privacy of a user's information. Such a system is ideal for areas with limited communication infrastructure.

There is need to develop more neural networks that are better suited for TinyML applications and that do not require the model to be optimised to a lightweight version for execution on an embedded microcontroller.

Chapter 3

Methodology

The goal of this project is to develop a system that can run a Machine Learning model that uses real-time sensor data on a microcontroller. This chapter will give an analysis of the user requirements to achieve this goal. Specifications of the user requirements and tests to validate them will be given. Furthermore, the system architecture and subsystem breakdown will be presented with justification for each design choice. The main purpose of this chapter is to give enough detail that will allow the work to be replicated.

3.1 User Requirements

An overview of the user requirements for the Environmental Monitoring and Forecasting System will be given. These functional requirements were derived from the deliverables given in Table 3.1 below. Specifications of the requirements and verification to show that the system meets the user requirements will also be explained.

Table 3.1: List of Project Deliverables

Milestone ID	Milestone Description
M001	Train the model in the cloud where high-power computational resources are available.
M002	Adapt the model to a lightweight environment. For instance, migrate from TensorFlow to TensorFlow Lite.
M003	Run the inference on a particular microcontroller such as through the Arduino library.
M004	Feedback of the inference should be uploaded to the cloud for analysis.

Table 3.2: User Requirements

Reqs ID	Description
UR001	Develop and train a Machine Learning model for predicting time series data.
UR002	Adapt the trained model to a lightweight model.
UR003	Adapted model should run on a low power device.
UR004	Adapted model should not deviate much from original model.
UR005	System should be cost effective.
UR006	Computation of adapted model should be in real-time.
UR007	The monitoring system should be user friendly.

3.1.1 Requirements Analysis

Table 3.3: Specifications

Spec ID	Description	Derived from
SP001	The Machine Learning model for predicting temperature should be trained on the cloud where there are enough computational resources.	UR001
SP002	The trained model should be optimised to a lite version with less memory.	UR002
SP003	The optimised model should be able to run on low power embedded microcontroller.	UR003
SP004	The optimised model should perform as expected as the original model with RMSE and MAE values of less than 3.	UR004
SP005	Hardware used to build the system should be cheap and accessible.	UR005
SP006	Execution of the model on the microcontroller should be fast, less than 10 milliseconds, using real-time temperature readings.	UR006
SP007	A screen/monitor will be used to display system build-up process and the environmental conditions.	UR007

Table 3.4: Specifications and Verification

Specs ID	Verification
SP001	Model will be trained with a training set after which testing, and validation sets will be used to test if the model produces accurate predictions.
SP002	The file size of the optimised model will be compared to the file size of the original trained model.
SP003	The optimised model will be deployed on a microcontroller for execution.
SP004	The optimised model will be executed to make temperature predictions and the accuracy of the predictions will be compared to the accuracy of the predictions of the original trained model.
SP005	Total cost of the hardware used should be under the R2,500 cost cap of the project.
SP006	Execution time of the inference of the optimised model on the microcontroller will be recorded and compared to the execution time of inference of the original trained model on a device with a more powerful processor.
SP007	As the system setup executes the process should be displayed on the screen. Upon completion of setup, current environmental conditions and the temperature prediction should be displayed on the screen.

3.2 Design Choices

This section is an overview of the overall Environmental Monitoring and Forecasting system design. An in-depth analysis of the design choices in hardware and software will be given with justification of each choice. The system's architecture will be broken down into different subsystems and an explanation of how these subsystems interact with each other will be provided.

3.2.1 System Architecture

Figure 3.1 shows the basics structure of the system. The microcontroller will be connected to a temperature sensor, humidity sensor and pressure sensor from which it will receive real-time readings of the environmental conditions as input. The hourly temperature readings will be used as input to the Machine Learning Model for it to forecast. The current environmental conditions and the predicted temperature will then be outputted on a display that will be connected to the microcontroller. From this system architecture,

the hardware chosen was derived.

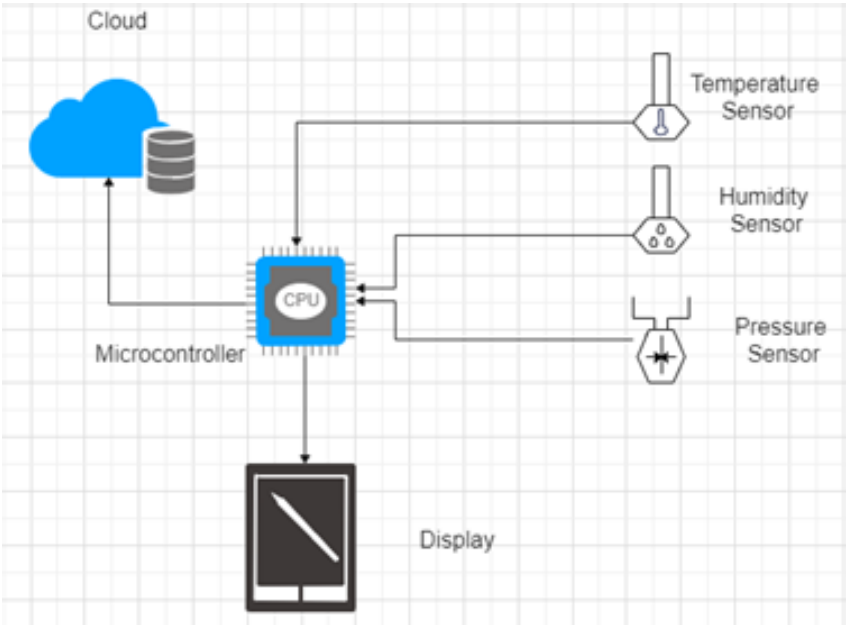

Figure 3.1: System Architecture

3.2.2 Hardware Used

Microcontroller

The first microcontroller option was the Arduino Nano 33 BLE Sense. This microcontroller is a 3.3V AI enabled board. It has a 64 MHz clock speed ARM Cortex-M4F processor with a Single-precision floating-point unit (FPU). It has a Bluetooth Low Energy Module and no Wi-Fi. The CPU has a flash memory of 1 MB and 256 KB SRAM [22]. This board has many sensors embedded on it, but the sensors of interest are the temperature sensor, humidity sensor and the pressure sensor. Making this microcontroller the first choice was based on related work with this project's application as indicated in the literature review. Many papers have used the Arduino Nano 33 BLE Sense in their TinyML applications and have indicated the microcontroller to be the most suitable option. Getting this microcontroller was however difficult due to its high demand. For this reason, an alternative microcontroller was selected.

Figure 3.2: Arduino Nano 33 BLE Sense

The second option was the ESP32-dev-module development board with ESP-WROOM-32. The ESP32 is one of few boards which are supported by software used to implement TinyML. It has an Xtensa single-core 32-bit LX6 microprocessor. The microprocessor has a clock speed of 240 MHz. For memory it has 448 KB of ROM and 520 KB of SRAM [15]. Unlike the Arduino Nano 33 BLE Sense, the ESP32 has no sensors embedded on its board. However the ESP32 has both Bluetooth and Wi-Fi modules.

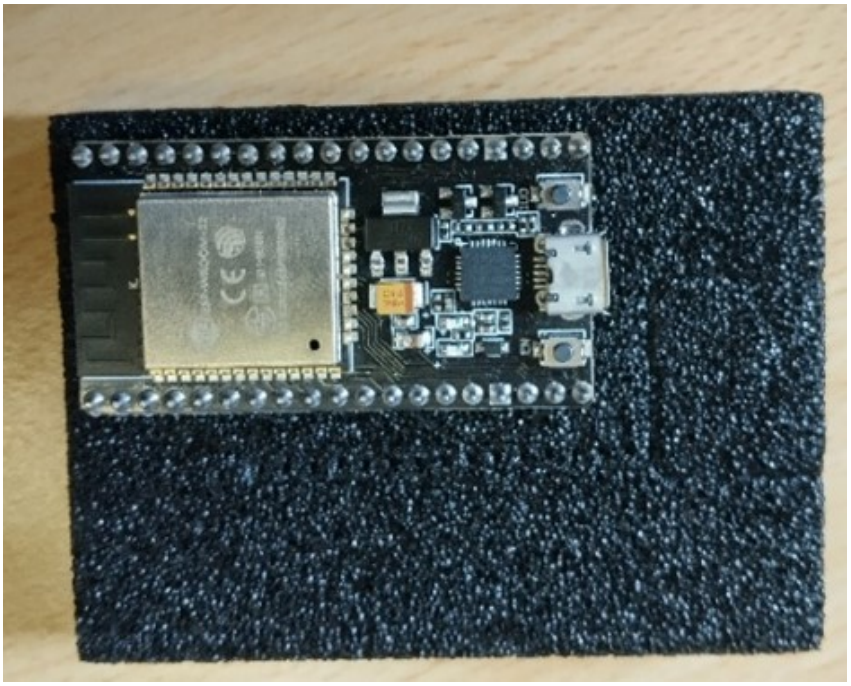

Figure 3.3: NodeMCU ESP32

Table 3.5: Comparison of Arduino Nano 33 BLE Sense and ESP32-dev-module

	Arduino Nano 33 BLE Sense	**ESP32-dev-module**
CPU	32-bit ARM Cortex-M4F	Xtensa 32-bit
Number of Cores	1	2
Flash Memory (MB)	1	0.448
SRAM (KB)	256	520
Clock Speed (MHz)	64	240
I2C	1	2
SPI	1	4
Sensors	IMU, temperature, humidity, pressure, Digital Proximity, Ambient Light, RGB and Gesture	None
Wireless	Bluetooth	Bluetooth and Wi-Fi

Sensors

The Arduino Nano 33 BLE Sense has embedded sensors therefore there is no need to connect external sensors to the board. The HTS221 is embedded onto the board. It contains the temperature and humidity sensors. Also embedded on the board is the LPS22HB sensor which measures the absolute pressure. However, the temperature sensors can get distorted when the microcontroller overheats.

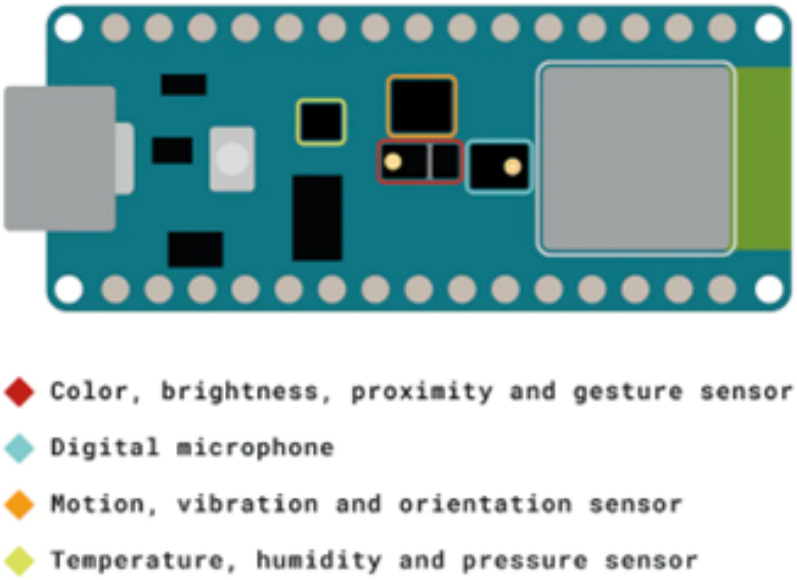

Figure 3.4: Temperature, Humidity and Pressure Sensors on the Arduino Nano 33 BLE Sense

Since the ESP32 does not have sensors embedded on it, external sensors were needed to be connected to the board using the I2C/SPI interfaces. Adafruit's BME680 was used. This board contains a temperature sensor, humidity sensor, pressure sensor and gas sensor. Volatile organic compounds (VOC) in the air change the resistance of the gas sensor. Gases such as Carbon Monoxide and alcohols such as ethanol are detected and used to measure the air quality.

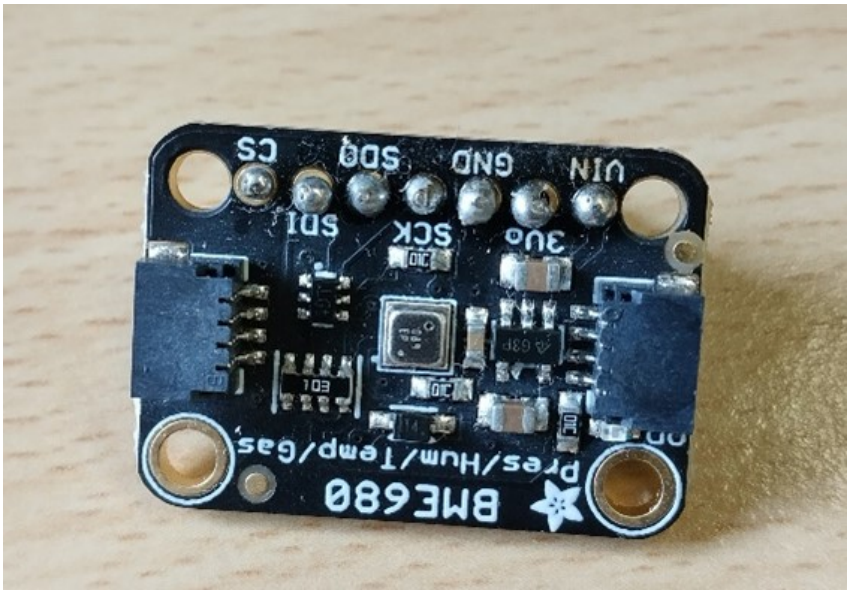

Figure 3.5: Adafruit BME680 Temperature, Humidity and Pressure Sensor

Table 3.6: Comparison of Arduino Nano 33 BLE Sense HTS221 Temperature and Humidity Sensor and the BME680 Temperature and Humidity Sensors

	HTS221 Temperature and Humidity Sensor	**BME680 Temperature Sensor**	**BME680 Humidity Sensor**
Voltage Range (V)	1.7 – 3.6	1.7 – 3.6	1.7 – 3.6
Operating Temperature (*C)	-40 – 120	-40 – 85	-40 – 85
Temperature Accuracy (*C)	±0.5 (@ 15 – 40 *C)	±1.0 (@ 0 – 65 *C)	-
Humidity Range (%)	0 – 100	-	0 - 100
Humidity Accuracy (%)	±3.5 (@ 20 – 80 oC)	-	±3
SPI	Yes	Yes	Yes
I2C	Yes	Yes	Yes

Table 3.7: Comparison of Arduino Nano 33 BLE Sense HTS221 Temperature and Humidity Sensor and the BME680 Temperature and Humidity Sensors

	LPS22HB Pressure Sensor	**BME680 Pressure Sensor**
Voltage Range (V)	1.7 - 3.6	1.7 - 3.6
Operating Temperature (*C)	-40 - 85	-40 - 85
Absolute Pressure (hPa)	260 - 1260	300 - 1100
SRAM (KB)	±0.1 (@ 0 - 65 *C)	±0.6 (@ 0 - 65 *C)
Absolute Pressure Accuracy (hPa)	±0.1 (@ 0 – 65 *C)	±0.6 (@ 0 – 65 *C)
I2C	Yes	Yes
SPI	Yes	Yes

3.2.3 OLED Display

The display used to output the environmental conditions and forecasted temperature was the Adafruit SSD1306. It is a 128x64 OLED display with an operating voltage of between 1.65V – 3.3V. The OLED display has four pinouts, VCC which takes in 3.3V – 5V, Ground (GND) pin, SCL and SDA pins. As indicated by the pins, the only interface supported is the I2C. Various microcontrollers are supported with it being compatible with the Arduino, ESP32, STM32, Raspberry Pi and many more. One of the biggest advantages of this display is its low power consumption. When it is fully lit it consumes 0.08W [24]. It does not have backlight.

A cheaper 128x32 of the same model was considered. However, the screen was too small to clearly display all the environmental conditions. This would go against the user requirement that states that the system should be user friendly (UR007).

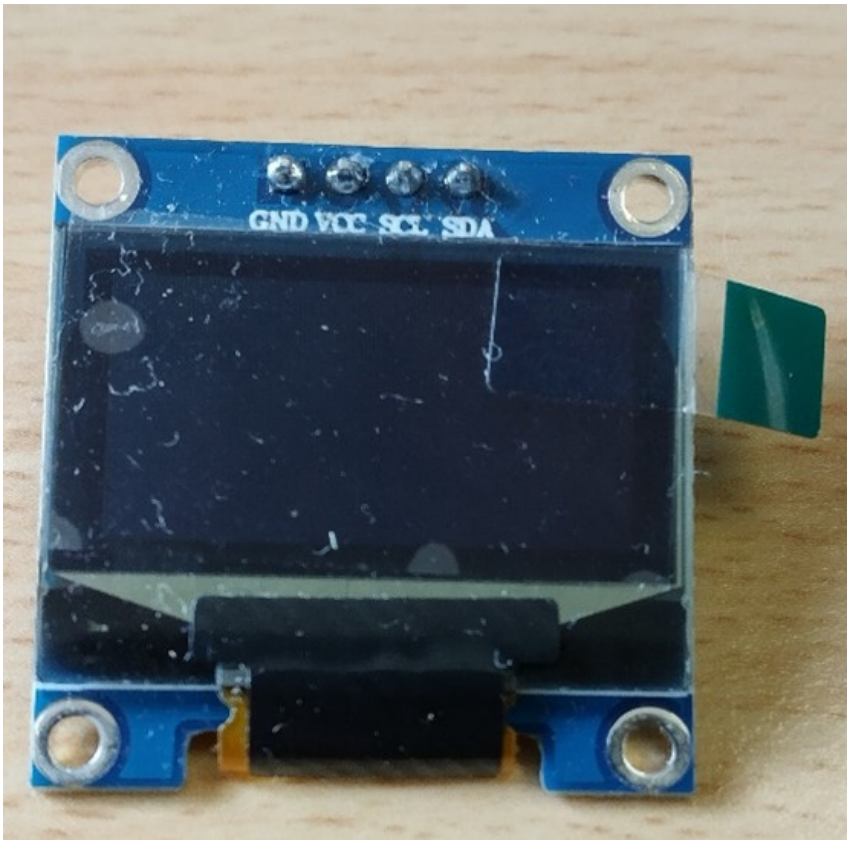

Figure 3.6: Adafruit SSD1306 OLED Display

3.2.4 Software Used

Python

To train the Time Series Forecasting Machine Learning model, Python on Google Colab was used. Python is the most used language for Machine Learning because of its readability and flexibility. It is also more suited for statistical data and data analysis. The main Python libraries used to develop the model are:

- **TensorFlow,** used to train and develop the neural network.
- **Keras,** a deep learning framework that uses a computers CPU and GPU to do fast computation and prototyping.
- **Pandas,** used to analyse complex data, organising, and filtering the data. It also allows extracting from other sources such as Excel.

- **NumPy,** suited to data science computations. It provides multi-dimensional arrays which allow operation of scientific calculations to be done easily.
- **Matplotlib,** helps with data visualisation by plotting data on graphs such as histograms, charts and scatter plots.

Google Colab is a platform where Jupyter Notebooks are run on the cloud. It provides free access to GPUs giving enough computational resources to train and develop Machine Learning models. There are several advantages of using Google Colab over Jupyter Notebook for Machine Learning applications. Google Colab has pre-installed Machine Learning libraries such as TensorFlow, Keras, and PyTorch making the setup simple. Work done on Google Colab is saved on the cloud making it accessible anywhere by any machine. Jupyter Notebook saves the work locally on the machine. This has an advantage of privacy. It is easier to collaborate on a project using Google Colab where multiple developers can work on the same project code.

Arduino IDE

Development of the boards was done using the Arduino IDE. The Arduino IDE uses the C++ programming language for development. Both the Arduino Nano 33 BLE Sense and ESP32 are compliant with the IDE and libraries for the respective boards are provided. Libraries for all the sensors are also available. The IDE is used to upload code onto the microcontrollers that will enable it to do the environmental monitoring by extracting readings from the sensors and load the Time Series Forecasting model that uses the real-time temperature readings to run the inference and predict the temperature.

Netron

This software is used as a visualiser for a machine learning model, neural network and deep learning. It displays the nodes in the internal structure of the model's neural network. Netron supports TensorFlow Lite, Keras, PyTorch and many other models.

ThingSpeak

ThingSpeak is an Internet of Things analytics platform. Sensor data is transmitted to the cloud. The data can be visualized and analysed. When the system is deployed, the actual temperature reading, humidity, pressure and the predicted temperature will be uploaded to this platform after each inference. The predicted temperature will be compared to the actual temperature reading to observe how accurate the prediction was. Analysis results of this data can be used to train a better model which will be deployed on the microcontroller.

3.2.5 Subsystem Breakdown

The whole system was divided into three subsystems to ensure that all the functional requirements (specifications) are met.

Model Training

This subsystem operates separately from the rest of the system. Training of the model is done on the cloud using GPUs and CPUs provided on Google Colab. The model is trained using Python and the TensorFlow and Keras libraries. Different Neural Networks, that is the Multilayer Perceptron, Convolutional Neural Network, and the Recurrent Neural Network (LSTM), will be used to train the model and tested on the Mean Absolute Error to find the most suitable for the model. After training the model, it will be converted to TensorFlow Lite. It will undergo post training quantisation during this conversion to a tflite file. The tflite file will finally be converted to a C byte array which will be deployed onto the microcontroller.

Sensing Subsystem

This subsystem consists of the temperature, humidity, pressure, and gas sensors used to take readings of the environmental conditions. The raw sensor data is provided as input to the whole system. The sensors are slaves in an I2C connection to the microcontroller. I2C offers simplicity in connecting multiple devices as it uses less wires as compared to SPI. Figure 14 shows the sensors that are on the Arduino Nano 33 BLE Sense. As

shown, the microcontroller has many in-built sensors, but the only sensors required for the system are the temperature, humidity, and pressure sensors.

Processing Subsystem

This subsystem consists of the ESP32. It is responsible for executing the code uploaded using the Arduino IDE. The microcontroller receives sensor data of the environmental conditions from the Sensing Subsystem. This data is processed and sent to the display as output. The temperature readings are used as input to the trained model which uses this 'historical' data to make predictions of the temperature. The microcontroller acts a master in the I2C connections to the sensors and the OLED display.

Display Subsystem

This subsystem consists of the Adafruit SSD1306 OLED display. It has the simple task of displaying the environmental conditions read from the sensors by the microcontroller. The display is also a slave to the microcontroller in an I2C connection.

3.3 Development Process

The system framework does not follow a linear path of development. This is because training a Machine Learning model is not a straight-forward process. Data has to be continuously analysed and processed to make it suitable for the application it is intended to be used in. Integrating the software with the hardware can give rise to incompatibilities which need to be attended to. The functional requirements may need to be adjusted way into the implementation. For this reason, an iterative development process is required. The development process selected was the Spiral model.

The Spiral model allows for continuous development and is flexible for projects with complex requirements which may need to change during the course of the project. With this project there might need to be a trade-off of accuracy when adapting the model to a lightweight version so that it is able to run on the microcontroller. The Spiral model is useful for when a prototype is needed in the development process. It allows for different iterative versions of the prototype as it is developed.

This Spiral model was designed with the objective of accomplishing the milestones mentioned in Table 3.1. The model is split into six sections to be followed for each development cycle, that is defining the problem, designing the solution, collecting information, analysing the information, evaluating the implemented solution and reporting and documenting the findings. Based on the findings, the problem can be redefined, and the cycle is repeated until all the milestones are met.

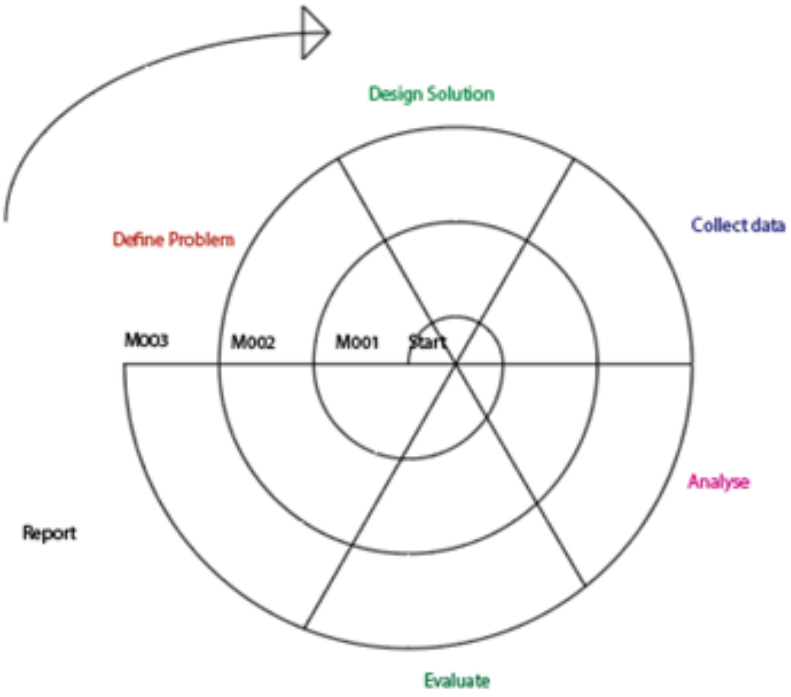

Figure 3.7: Spiral development model used to meet the project milestones

3.3.1 Development Timeline

To be able to meet all the targets of the project within the given deadline of the 6th of November 2022, tasks were broken down and their completion scheduled to meet the deadline. The tasks involved:

1. Understanding the problem statement. This involved getting familiar with the project's topic by researching the topic's background and its practical applications. This helped to clearly define the objectives of the project which are within the problem statement. The scope of the project was then outlined.

2. Theory. Understanding the relevant theory related to the project. This was done by going through the main concepts and indicating a personal interpretation of how

the concepts were understood from other researchers' work and textbooks.

3. Literature review. Key words and/or phrases were used to find papers relevant to the project. The main sources of the literature were the IEEE Explore and Google Scholar. The aim of the literature review is evaluating previous related work done and validate or criticise the methods or results presented.

4. Design and planning. This involved justification of the approach used to solve the problem statement. In the design, details on how the solution to the problem statement were explicitly given with the aim of making it easier for someone else to replicate this work.

5. Simulation and analysis of data. This mainly involves testing the design to ensure it aligns with solving the problem statement. The objectives of the tests were presented and how the tests were set up.

6. Results, discussion, and conclusion. The results obtained from the tests were presented. Explanation on what they mean and reference to the issues addressed by the problem statement was outlined. Correlation of these results to the literature presented was done.

3.4 Steps of Implementation

Training Model

The Time Series Forecasting model was trained using Google Colab. To use Google Colab, a Google account is required. The dataset used was obtained by connecting the BME680 sensor to the ESP32 board. The system was placed outside for 24 hours to collect the temperature, humidity, pressure and air quality data. A channel was created on ThingsSpeak, on the cloud, where the sensor data was uploaded after every 10 minutes. After the 24 hours of collecting the data, the 24-hour data was replicated to produces a dataset equivalent to seven years of daily readings.
This was done because climate data for Cape Town could not be acquired and for the model to work, weather data representative of the region of application was needed.

	created_at	entry_id	Temperature (degC)	Humidity (%)	Pressure (hPa)	Air Quality (Ohm)
0	2022-11-02 01:00:00	1	22.79945	52.08028	1018.10999	170858
1	2022-11-02 01:10:00	2	21.54753	49.60780	1018.03003	234857
2	2022-11-02 01:20:00	3	20.93671	48.81311	1017.75000	238414
3	2022-11-02 01:30:00	4	19.51950	51.14481	1017.65997	255752
4	2022-11-02 01:40:00	5	19.29297	52.39877	1017.48999	244769

Figure 3.8: Showing first 5 entries in the dataset

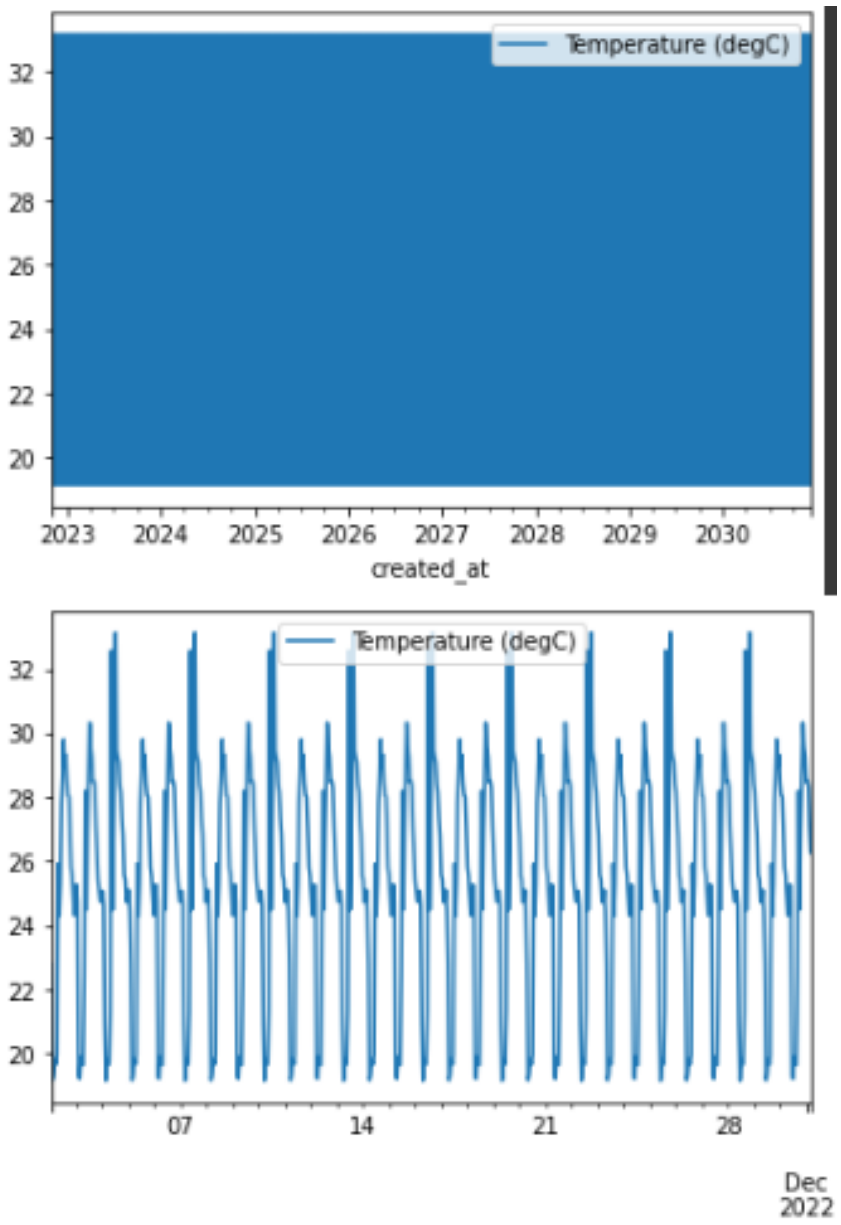

Figure 3.9: Temperature data equivalent to 7 years

The data was refined to account for 1-hour intervals starting at the first entry. From there, the 1-hour intervals were obtained by extracting 6 timesteps. This refined data was then used to train the model.

created_at	created_at	entry_id	Temperature (degC)	Humidity (%)	Pressure (hPa)	Air Quality (Ohm)
2022-11-02 01:00:00	2022-11-02 01:00:00	1	22.79945	52.08028	1018.10999	170858
2022-11-02 02:00:00	2022-11-02 02:00:00	7	19.17245	53.46775	1017.32001	235359
2022-11-02 03:00:00	2022-11-02 03:00:00	13	19.33430	52.61555	1016.89001	229636
2022-11-02 04:00:00	2022-11-02 04:00:00	19	19.81102	52.53099	1016.73999	220595
2022-11-02 05:00:00	2022-11-02 05:00:00	25	19.65832	53.53024	1016.76001	224338

Figure 3.10: Refined data at 1-hour intervals

One of the most important components of the time series weather data is its daily and yearly periodicity. To test for this periodicity, the "created_at" string column was converted into seconds:

```
df.index = pd.to_datetime(df['Date Time'], format='%d.%m.%Y %H:%M:%S')
timestamp_s = df.index.map(pd.Timestamp.timestamp)
```

Listing 3.1: Converting timestamp from string to seconds

After the conversion, the number of seconds in a day and the number of seconds in a year were calculated. The value for each daily time stamp was calculated and stored in an array which was used to plot the sine and cosine graphs to prove the periodicity of the data.

```
df['Day sin'] = np.sin(timestamp_s * (2 * np.pi / day))
df['Day cos'] = np.cos(timestamp_s * (2 * np.pi / day))
```

Listing 3.2: Loading periodic function of time series data

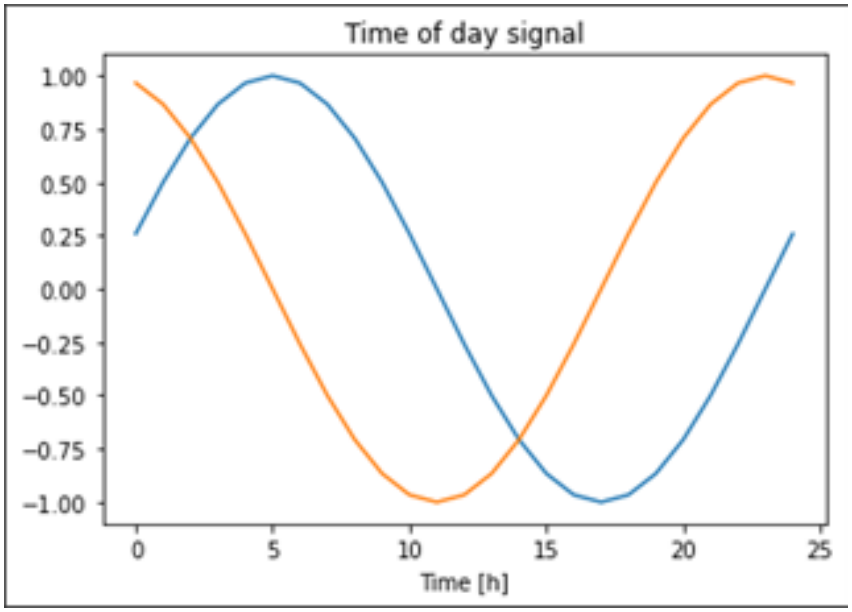

Figure 3.11: Sine graph showing periodicity of data

Once the data was refined, it was split to a training set used to train the model, a testing set used to test the trained model and a validation set used to further test the accuracy of the trained model. The split ratio was 70% for the training set, 20% for the validation set and 10% for the testing set. The statistical information of the refined data is given below:

```
count    70956.000000
mean        25.319663
std          3.562352
min         19.163310
25%         24.301900
50%         25.428290
75%         28.300790
max         33.130350
Name: Temperature (degC), dtype: float64
```

The model was trained using three different Neural Networks, the Mulitilayer Perceptron, Convolutional Neural Network and LSTM (Recurrent Neural Network). Performance of these neural networks was compared based on their mean absolute errors.

Migrating from TensorFlow to TensorFlow Lite

Once the model was trained, it was then converted to TensorFlow Lite which is a lightweight version of the original trained TensorFlow model. Different post training optimisation settings can be implemented to reduce model size which reduces the amount of processing, power and CPU latency when running inference on the microcontroller. A tflite file is generated.

```
converter = tf.lite.TFLiteConverter.from_keras_model(model1)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS,
   tf.lite.OpsSet.SELECT_TF_OPS]
tflite_model = converter.convert()

# Save the model.
with open('dense_modelr.tflite', 'wb') as f:
  f.write(tflite_model)
```

Listing 3.3: Converting trained model to TensorFlow Lite

Convert to C-byte Array

To be able to deploy the model onto the Arduino Nano 33 BLE Sense/ESP32 and run the inference, the tflite was converted to a C-byte array which can be understood by the device. The tflite was converted to a C header file using the following unix command **xxd -i converted_model.tflite > model_data.h**.

Arduino IDE Setup

The Arduino IDE Desktop version was installed from the Arduino website. The IDE was setup to enable the programming of the Arduino Nano BLE Sense. The Arduino Mbed OS Nano Core for the Arduino Nano 33 BLE Sense and esp32 by Espressif Systems were installed by selecting **Tools menu>Boards>Boards Manager** in the IDE. After the installation is complete, the board type and port need to be selected. This can be done under the Tools menu.

The next step is setting up the libraries that enable reading of the sensors. These libraries were added by going to the **Sketch menu>Include Libraries>Manage Libraries** and searching for the sensors of interest and installing them. After everything was setup, a new sketch was started and the code was written, model loaded and uploaded to the microcontroller.

Below is the pseudo code of the libraries imported to configure the sensors, I2C and SPI intefaces, and the TensorFlow Lite Micro, EloquentTinyML library, that will load the model for inference.

```
//import libraries for the sensors and OLED screen, and I2C interface
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h> // for the sensors
#include "Adafruit_BME680.h"    // BME680 library
#include <Adafruit_GFX.h>      //OLED graphics
#include <Adafruit_SSD1306.h>   // OLED library
// Import the required TensorFlow modules.
#include <EloquentTinyML.h>
#include <eloquent_tinyml/tensorflow.h>
//WiFi module
#include <WiFi.h>
```

```
13 #include "ThingSpeak.h"
```

Listing 3.4: Libraries needed for configuring sensors and loading TensorFlow Lite model

Final System Setup

Once the model is trained and adapted for inference on the microcontroller and the software is compatible with the hardware, the microcontroller, sensors, and the OLED display are going to be mounted on a board. The system will be powered by a 9V battery. This will make it mobile and can be easily installed in a greenhouse.

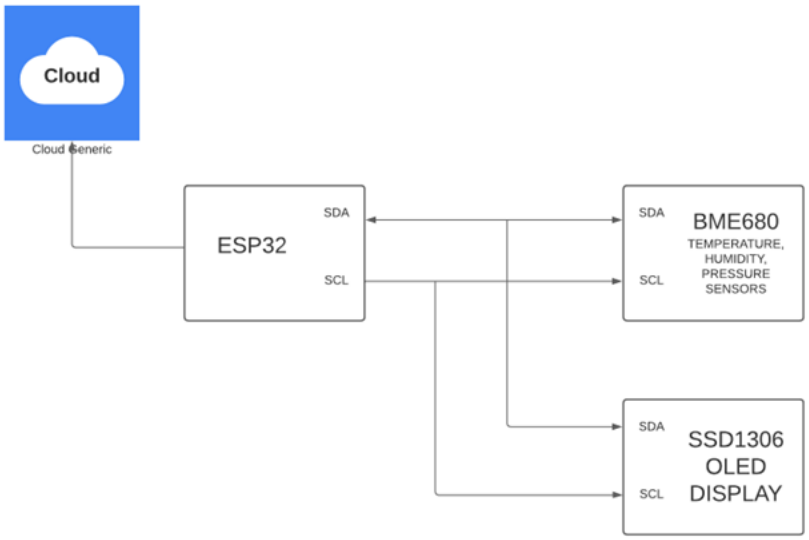

Figure 3.12: I2C connection between ESP32 (master) and BME680 Sensor (slave) and SSD1306 OLED Deisplay (slave)

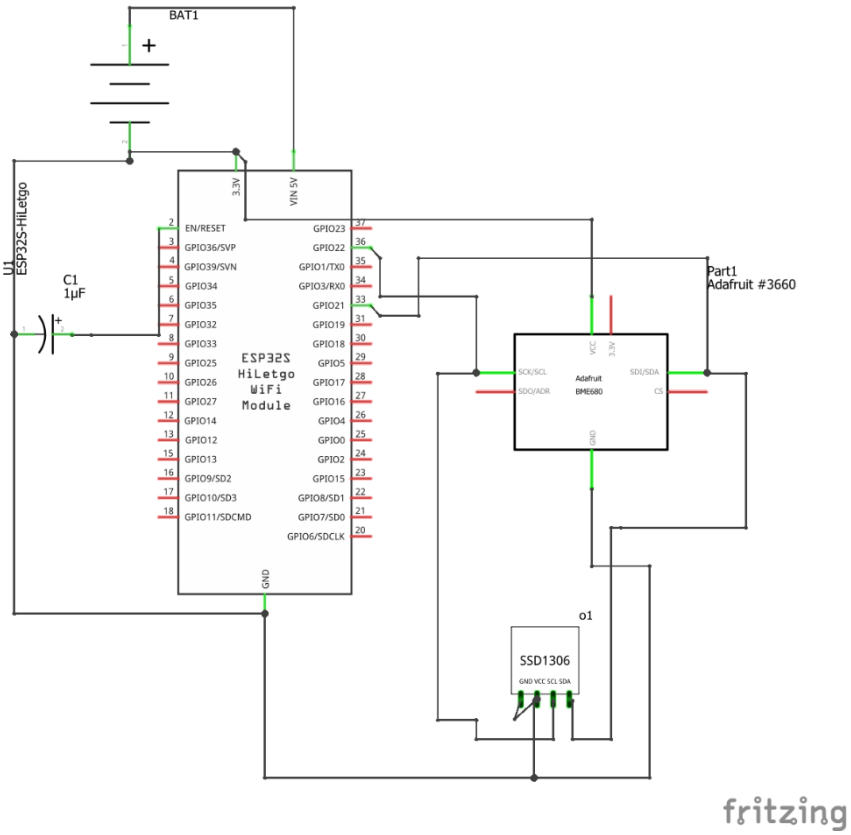

Figure 3.13: Schematic of ESP32, BME680 and SSD1306 connections

3.5 Constraints and Limitations

Internal Storage

The Arduino Nano 33 BLE Sense has very low storage of 1 MB and the ESP32 has a storage of 448 KB. For this reason, the size of the model is very crucial because if it is not optimised it cannot be uploaded and run on the target device. For this reason, things like data types and optimisation techniques like normalisation and post training quantisation are needed to adapt the model for a restricted device.

Processing Power

The Arduino Nano 33 BLE Sense and ESP32 have a CPU with SRAM of only 256 KB and 520 KB and clock speed of 64 MHz and 240 MHz respectively, Table 3.5, which are very low processing powers. Machine Learning models are normally executed on accelerated

GPUs with more computational resources.

Power Consumption

The Arduino Nano 33 BLE Sense and ESP32 are low power devices, and the system will be powered by a battery. Power optimisation will need to be implemented in the code to put the device in low power mode when inference is not taking place.

3.6 System Workflow

This section describes how a TinyML, (TensorFlow Lite Micro), application is executed and how this was implemented in the system. Pointers for the model, error reporter, input and output tensors and the interpreter are initialised. The amount of memory that will be used by the input and output tensors and the model's operations is reserved. After the variables are initialised, the error reporter is setup. When an error occurs during setup or inference the error reporter will flag the error. The model is loaded from the C byte header file. This process involves checking version compatibilities between the library and the model, building an interpreter used to allocate memory for the model's input and output buffers. Once the setup is complete, the input is fed into the model and the model is invoked to produce an output.

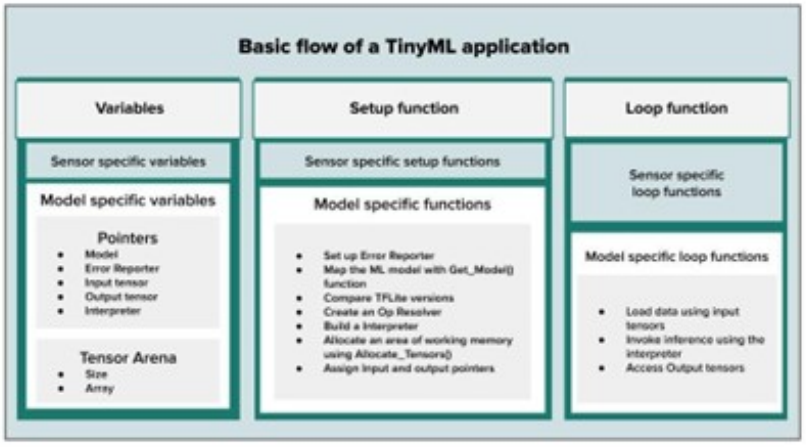

Figure 3.14: Workflow of TinyML application

Figure 3.15 below shows the workflow of the whole Environmental Monitoring and Temperature Forecasting system for the greenhouse. The sensors and the OLED display are configured

in the same Setup function that the model is loaded, as mentioned above. When the setup process is complete, a timer is started. The system is given five hours to load hourly temperature data that will be used for inference. After the loading process is completed, inference occurs and the actual temperature, humidity, pressure, and predicted temperature readings are displayed on the OLED. These values are uploaded to the cloud for analysis, whose data will be used to develop a better model. The system will go to a low power mode for an hour after which the temperature data used for inference is updated and inference occurs again. From this point the cycle repeats.

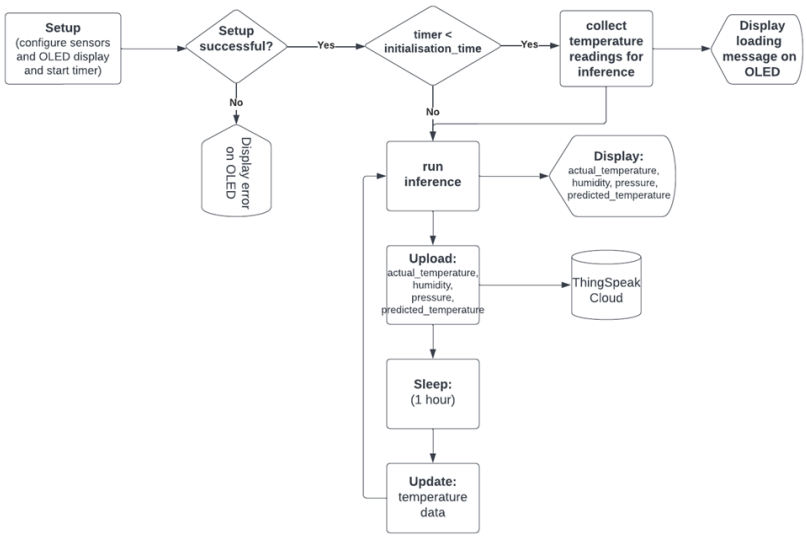

Figure 3.15: Workflow of Environmental Monitoring and Temperature Forecasting System

Chapter 4

Results

This Chapter gives a description of how the experiments to ensure that the acceptance tests in Table 3.4 were met. These tests were designed to make sure that the system does not deviate from its objectives and that the prototype operates as expected. The results of the tests and analysis of these results are then later given, commenting on whether the results came out as expected and justifying why they did not come out as expected.

4.1 Experiment Setups

4.1.1 Subsystem Tests

Processor: ESP32 Test

The first test was to power up the microcontroller using a micro-USB cable. This test was used to check if the red LED lights up indicating that the microcontroller is on. Success of the test confirms that the 3.3V on-board regulator works.

The next test was to test the serial port connection and how to upload code onto the board. The blink test was used, blinking an LED. An LED was connected to GPIO pin 23. The pin was set to high and low with a one second delay between the changes to turn the LED on and off. The pseudo code and circuit connection are shown below.

```
1 const int ledPin = 23;  // ledPin refers to ESP32 GPIO 23
```

```
void setup() {
  pinMode(ledPin, OUTPUT); // initialize digital pin ledPin as an
   output.
}
void loop() {
  digitalWrite(ledPin, HIGH);   // turn the LED on (HIGH is the voltage
    level)
  delay(1000);                  // wait for a second
  digitalWrite(ledPin, LOW);    // turn the LED off by making the
   voltage LOW
  delay(1000);                  // wait for a second
}
```

Listing 4.1: Blink Test Code

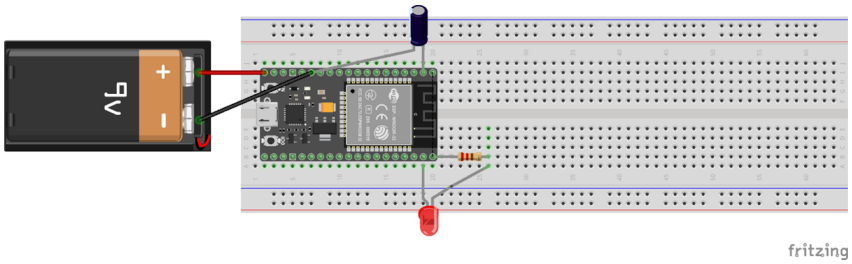

Figure 4.1: Blink Test Circuit

Sensors: BME680 Test

To test that the sensors are working, the BME680 which has the temperature, humidity and pressure sensors was connected to the ESP32. As mentioned in the previous section on how to setup the Arduino IDE Setup, the Adafruit BME680 Library and the Adafruit Unified Sensor Library were installed. These libraries were needed to configure the hardware functionality. After the libraries were installed, the example code used to test the hardware was opened in the Arduino IDE. This was done by going to the **File>Examples>Adafruit BME680 Library>bme680test**. The code was compiled and uploaded to the ESP32, and the results were observed. Success would be confirmed by viewing the sensor reading on the Arduino IDE Serial Monitor. The touch test was used to test the temperature sensor. When sensor was touched, a temperature increase was expected. Overall success of this test would also confirm that the I2C interface is functional.

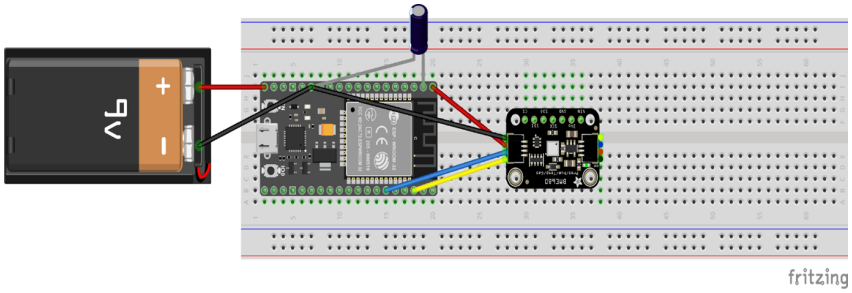

Figure 4.2: Sensor Test Circuit

Display: SSD1306 OLED Display

To test that the display was working, the SSD1306 OLED was connected to the microcontroller using the I2C interface. The Adafruit_SSD1306 and Adafruit_GFX libraries were installed. One of the example codes in the Arduino IDE was uploaded onto the ESP32 board to test the functionality of the display. The example used was from **File>Examples>Adafruit SSD1306>ssd1306_128x64_i2c**.

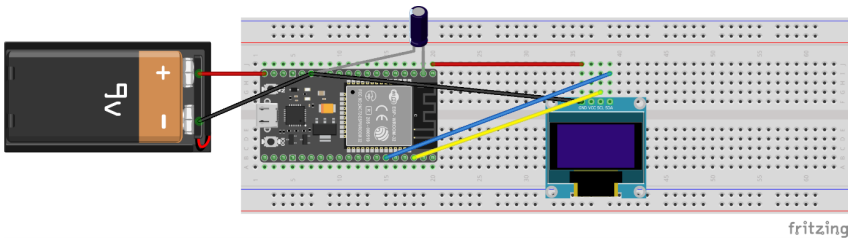

Figure 4.3: Circuit for OLED Display Test

4.1.2 Machine Learning Model Performance Tests

The model was trained using three different Neural Networks in Google Colab, the Multilayer Perceptron, Convolutional Neural Network, and the LSTM, a Recurrent Neural Network. The models were trained using 30-minute and 1-hour datasets to test how the model performed with each dataset. Tested was the execution time of these models to observe the amount of processing each requires. The quality of the predictions, that is how much the predicted temperature deviates from the actual temperature, measured using the Root Mean Square Error (RMSE), Mean Absolute Error (MAE) and accuracy metrics was also tested. The goal was to find the best performing Neural Network.

4.1.3 TensorFlow Lite Model Performance Tests

For each machine learning model trained using the three different Neural Networks and the 30-minute and 1-hour datasets. The models were optimised and converted to TensorFlow Lite. An interpreter object was created to get details of the model's shape, allocate memory for the input and output data of the model, and invoke the model for inference. Accuracy is the only metric that can be used to test the prediction quality. Comparisons were made of the accuracy of each converted model to the accuracy of the original model.

Figure 4.4 below shows the system deployed in the environment to read the temperature and make predictions. The cardboard box was to protect the devices from the sunlight to prevent overheating and reduce distortion of the temperature readings. The system was deployed for 10 hours from 10:00 am – 8:00 pm. The predictions were made for 15 – minutes into the future and the actual and predicted temperatures were uploaded to ThingSpeak.

Figure 4.4: System deployed into the environment to take temperature readings and make forecasts.

4.1.4 Accuracy vs Constraint

Effects of normalised and unnormalized data on the prediction quality of the trained models was tested. The metrics used were the Root Mean Square Error, Mean Absolute Error, and Accuracy. This effect was also tested on the converted TensorFlow Lite models to observe how there are affected.

4.1.5 Power Consumption Test

4.1.6 Execution Time

Efficiency in executing the model is crucial as the sensor data is collected in real time. Execution time is depended on the processing power of the microcontroller and the model's complexity. To test the speed at which the model is executed, the time taken to make a prediction by the ESP32 and the Arduino nano 33 BLE Sense was measured and compared. The in-built micros() timing function in Arduino IDE was used from the time the model was fed the input to the time the output was produced. With the ESP32 having the stronger processor, it is expected that its execution speed would be faster than the Arduino.

```
unsigned long t1 = micros();  // start timer
float predicted_temp = tf.predict(temp_readings);  // input loaded into
    model for inference
unsigned long t2 = micros();  // stop timer
```

Listing 4.2: Execution Time of running inference

4.2 Experimental Results

4.2.1 Subsystem Test Results

The red LED on the ESP32 confirms that the board is receiving power and that it is on. As shown in Figure 4.5, the LED is blinking confirming the success of uploading the code and the functionality of the GPIO pins. The serial connection between the PC and the board operated as expected.

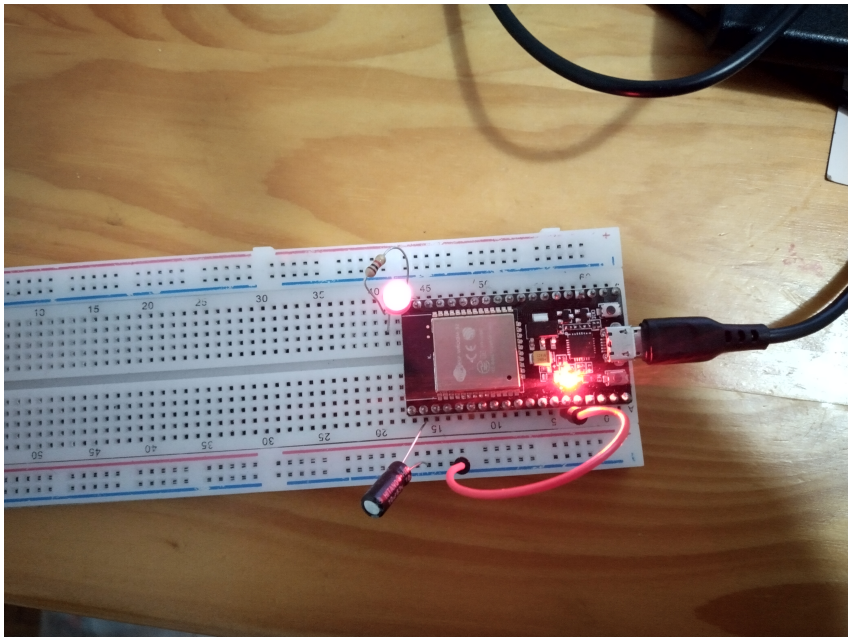

Figure 4.5: Blink Test

Figure 4.6 shows the output observed from the serial monitor of the Arduino IDE when BME680 test code was uploaded to the ESP32. It shows readings obtained from the temperature, humidity, pressure, and gas sensors on the BME680. The touch test was done on the temperature sensor to observe if temperature would increase if the sensor was touched. The temperature increased as expected confirming functionality of the temperature sensor.

```
Temperature = 25.69 *C
Pressure = 1010.97 hPa
Humidity = 64.20 %
Gas = 115.47 KOhms
Approx. Altitude = 18.92 m

Temperature = 25.77 *C
Pressure = 1010.98 hPa
Humidity = 64.30 %
Gas = 115.23 KOhms
Approx. Altitude = 18.83 m

Temperature = 26.75 *C
Pressure = 1011.02 hPa
Humidity = 65.46 %
Gas = 110.46 KOhms
Approx. Altitude = 18.92 m

Temperature = 27.97 *C
Pressure = 1011.02 hPa
Humidity = 65.95 %
Gas = 104.56 KOhms
Approx. Altitude = 19.25 m
```

```
Temperature = 27.97 *C
Pressure = 1011.02 hPa
Humidity = 65.95 %
Gas = 104.56 KOhms
Approx. Altitude = 19.25 m

Temperature = 29.00 *C
Pressure = 1010.99 hPa
Humidity = 65.96 %
Gas = 98.90 KOhms
Approx. Altitude = 19.25 m

Temperature = 29.81 *C
Pressure = 1010.99 hPa
Humidity = 65.82 %
Gas = 94.35 KOhms
Approx. Altitude = 18.92 m

Temperature = 30.44 *C
Pressure = 1011.01 hPa
Humidity = 65.66 %
Gas = 93.07 KOhms
```

Figure 4.6: Sensor Test and Touch Test results

4.2.2 Machine Learning Model Performance Results

Model Performance: Training Validation

Figure 4.7 below shows the root mean squared errors (RMSEs) of the model's predictions on the training and validation set during model training. The model was trained using 20 epochs which is the number of training iterations. As shown the model is a "Good Fit" with the validation RMSE is lower than the training's RMSE. This shows the ability of the model to make good predictions when it is fed new data, away from the training set which it is used to. This eliminates the worry of Overfitting which happens when the model picks up every detail of the training set such that it negatively affects its performance on new data. These results also validate that the algorithms used to train the model are fit for this application. If this was not the case, underfitting would occur where there are no improvements, that is decrease in RMSE value, during training and will obviously not perform well with new data.

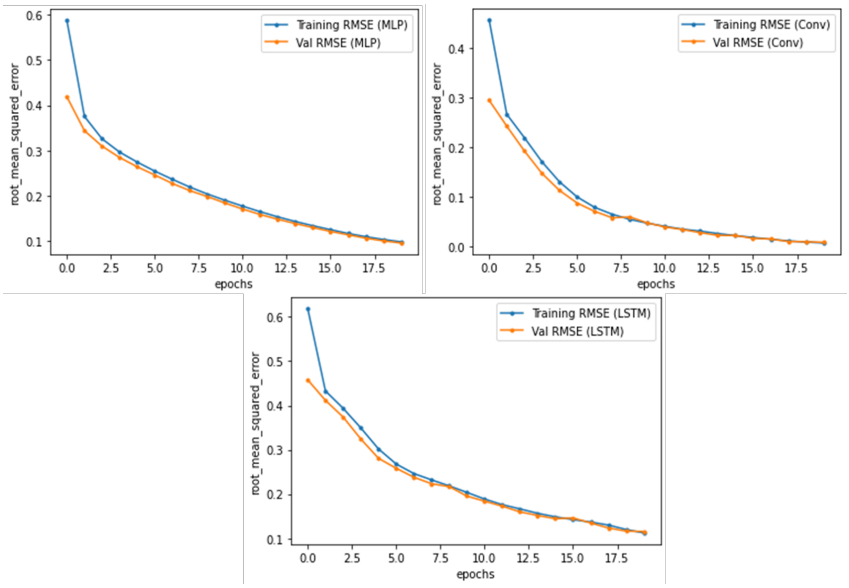

Figure 4.7: Validating Trained Models

Model Performance: 1-hour vs 10-minute dataset

Figure 4.8 below shows the results of the validation test after training the MLP, CNN, and LSTM models using the 1-hour datasets. The predicted and actual temperature values are plotted on the same graph. This is a visual presentation of how close the predictions are to the actual values. As expected, the three graphs show that predictions are quite close to the actual temperature values.

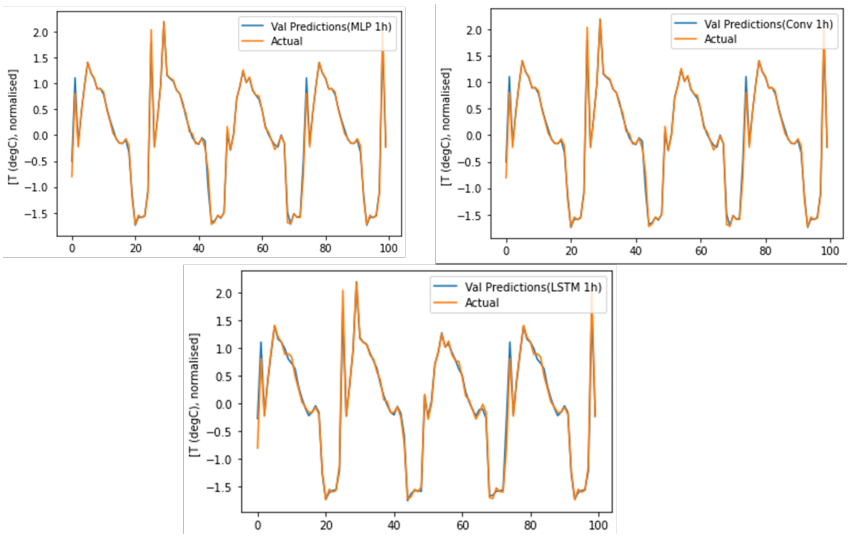

Figure 4.8: Validation test results for the MLP, CNN, and LSTM models for the normalised 1-hour dataset

Figure 4.9 shows the visual presentation of the validation test of the same three models trained by the 10-minute dataset. The predicted temperature plot is almost identical to the actual temperature plot. By observation, it is difficult to conclude whether the models trained by the 10-minute dataset give better predictions than the models trained by the 1-hour dataset.

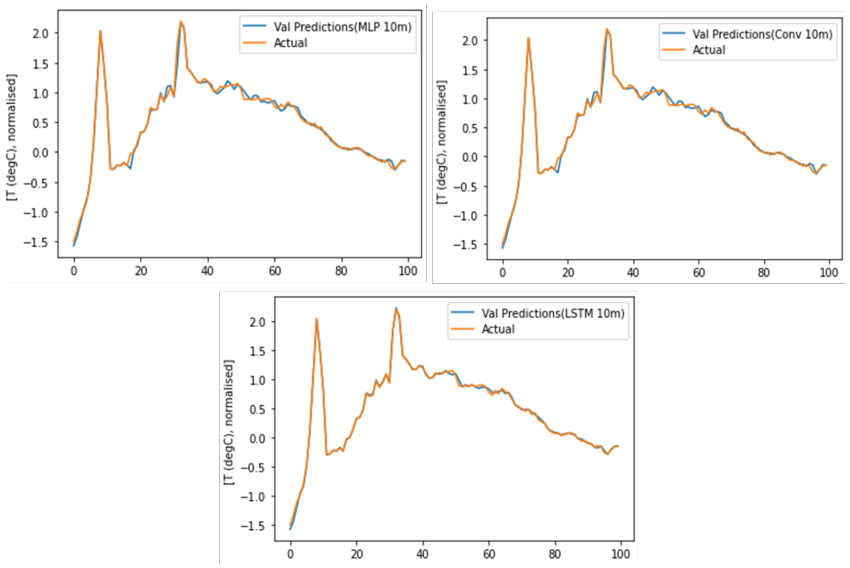

Figure 4.9: Validation test results for the MLP, CNN, and LSTM models for the normalised 10-minute dataset

Figure 4.10 and Figure 4.11 below show the root mean squared and mean absolute errors of the MLP, CNN, and LSTM trained models using both the 1-hour and 10-minute normalised datasets. As shown, the models trained by the 10-minute dataset perform better than the models trained by the 1-hour dataset as they have a smaller root mean squared and mean absolute errors. It is expected because the 10-minute dataset has more datapoints used to train the models than the 1-hour dataset, therefore for a model to perform well and make more accurate predictions, a larger dataset is preferred.

From these graphs, the best performing model can be derived. The CNN was the model that gave the most accurate predictions. This result was not expected as CNN models are mostly used in image classification applications. The LSTM model was expected to perform better since it is considered a “legacy” for time series forecasting with its temporal dependence feature. This feature is admirable for this application since temperature changes follow a trend during the course of the day, depending on the season, (periodicity).

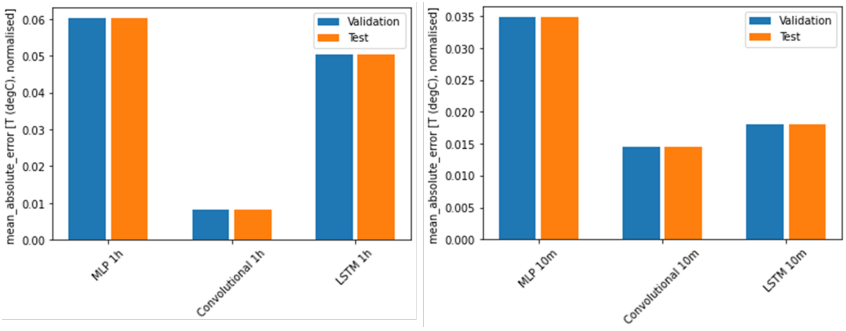

Figure 4.10: Mean Absolute Error of 1-hour dataset (left) and 10-minute dataset (right) for the MLP, CNN. and LSTM models from the Validation and Test sets

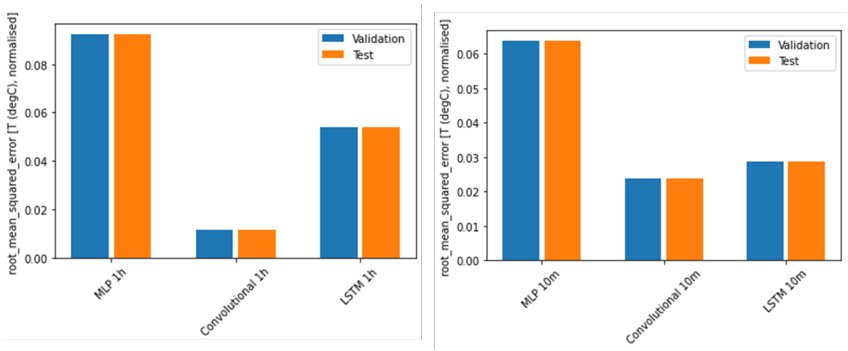

Figure 4.11: Root Mean Squared Error of 1-hour dataset (left) and 10-minute dataset (right) for the MLP, CNN. and LSTM models from the Validation and Test sets

Model Performance: Normalized vs Unnormalized

Normalisation puts data onto the same scale/distribution. This helps improve the performance of a machine learning model and makes training faster. The dataset was standardised. The distribution of the values was rescaled, making the mean zero and the standard deviation 1. The standardisation equation was $t' = \frac{t-\mu}{\sigma}$: where t′ - is the normalised temperature; t – is the temperature; μ - is the mean temperature; σ – is the standard deviation.

Figure 4.12 - Figure 4.16 show the comparison of the RMSE and MAE of normalised and not normalised data. The results show that the RMSE and MAE of all models is lower for normalised data. Normalising data improves the performance of the models hence they make more accurate temperature predictions. For normalised data, the CNN model performs the best and the LSTM model give the best predictions when the data is not normalised.

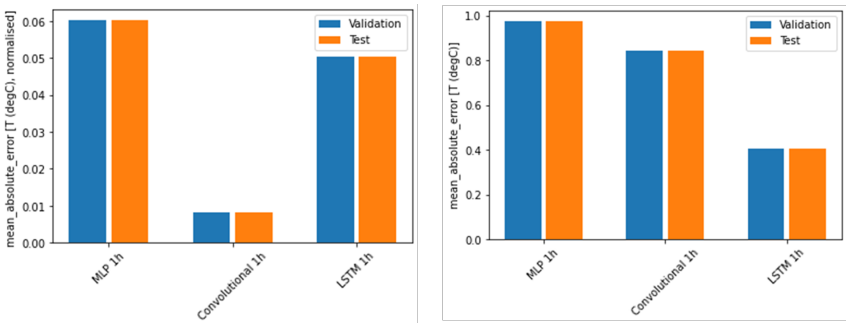

Figure 4.12: Mean Absolute Error Normalised (left) vs Not Normalised (right) for 1-hour dataset from the Validation and Test sets

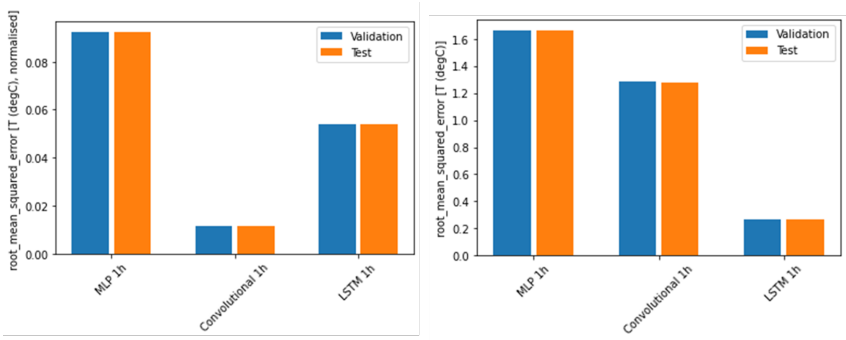

Figure 4.13: Root Mean Squared Error Normalised (left) vs Not Normalised (right) for 1-hour dataset from the Validation and Test sets

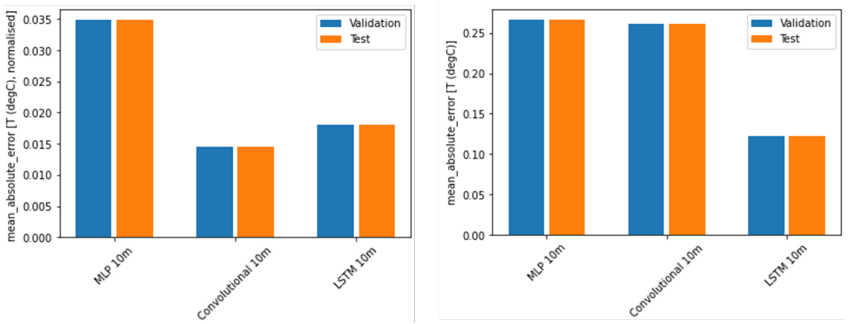

Figure 4.14: Mean Absolute Error Normalised (left) vs Not Normalised (right) for 10-minute dataset from the Validation and Test sets

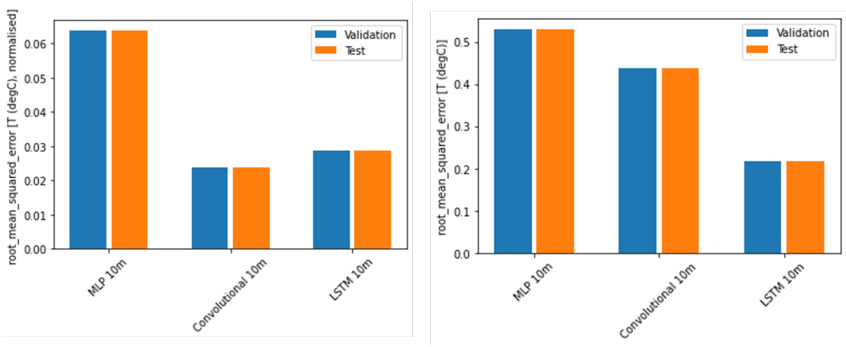

Figure 4.15: Root Mean Squared Error Normalised (left) vs Not Normalised (right) for 10-minute dataset from the Validation and Test sets

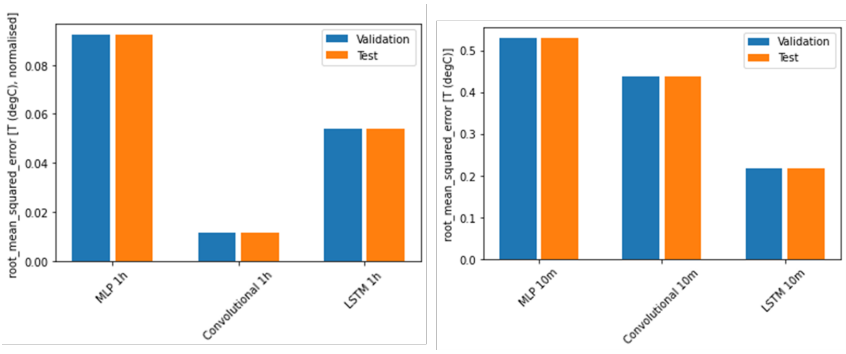

Figure 4.16: Root Mean Squared Error Normalised (left) for 1-hour dataset vs Not Normalised (right) for 10-minute data set from the Validation and Test sets

4.2.3 TensorFlow Lite Model Performance Results

Figure 4.17 below show the converted TensorFlow Lite models, tflite files generated, observed using Netron. The C-byte array header files generated from the tflite files of these models were used to deploy the models onto the ESP32 for inference.

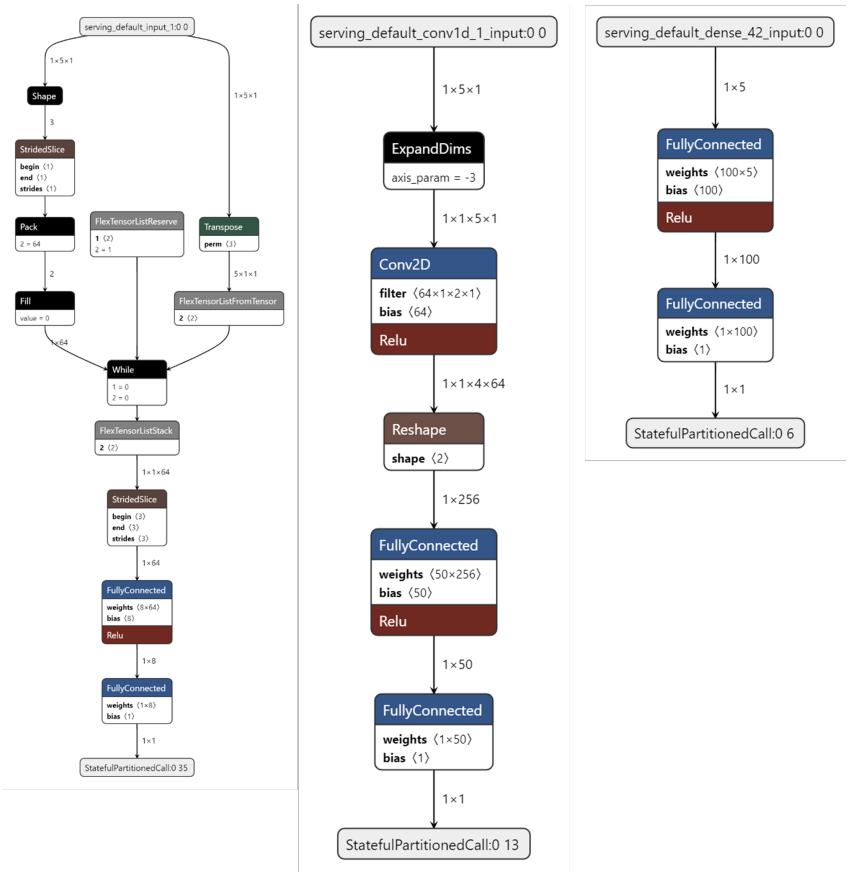

Figure 4.17: Tensorflow Lite LSTM model (left), CNN model (middle), and MLP model (right)

As observed, the LSTM model has the most complex structure with more operations needed to run the model. Within the layers and nodes of the model, there are operations that use the datatype int32. Within the TensorFlow Lite for Microcontroller libraries, only float32, float16, and int8 data types are supported. Therefore, the LSTM model could not be invoked. The error displayed on the Arduino IDE is shown in Figure 4.18 below.

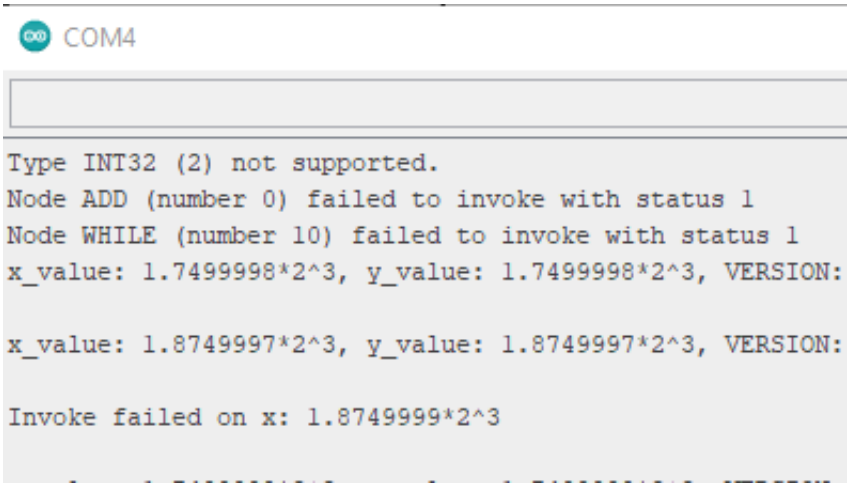

Figure 4.18: Error when LSTM model was invoked on the ESP32

The CNN model looks less complex than the LSTM model. For inference of the deployed TensorFlow Lite model to occur, it should be run through the interpreter as described in the system workflow in the previous chapter. The interpreter has a static graph, [26], within which it has subgraphs which are functions that contain the operations needed to execute the model in their order of execution. However, the TensorFlow Lite for Microcontrollers library only supports models with no more than a single subgraph containing the operations that are supported by the library [27]. Figure 4.19 below shows the error raised when the model was deployed onto the ESP32.

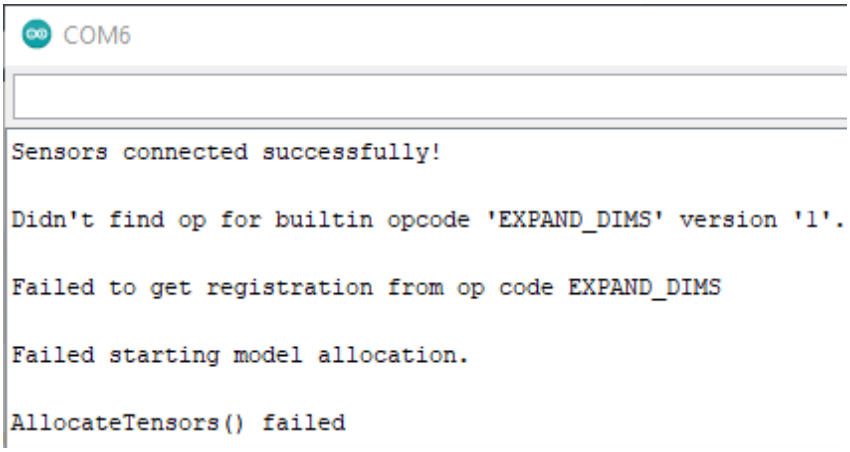

Figure 4.19: Error when CNN model was invoked on the ESP32

The MLP model is the simplest with only two layers. The model only has a single subgraph that has a few operations within it to execute the model. Deployment of the MLP on the ESP32 was successful and the interpreter ran the inference. Figure 4.20 shows the results of the test. The predictions do not deviate much from the actual temperature with the largest offset being approximately -3*C. It is observed that the

model takes about one and a half hours to grasp the trend of the temperature change after which predictions become more accurate. The calucated RMSE and MAE values were 1.45 and 1.3 respectively, (See Appendix A.2).

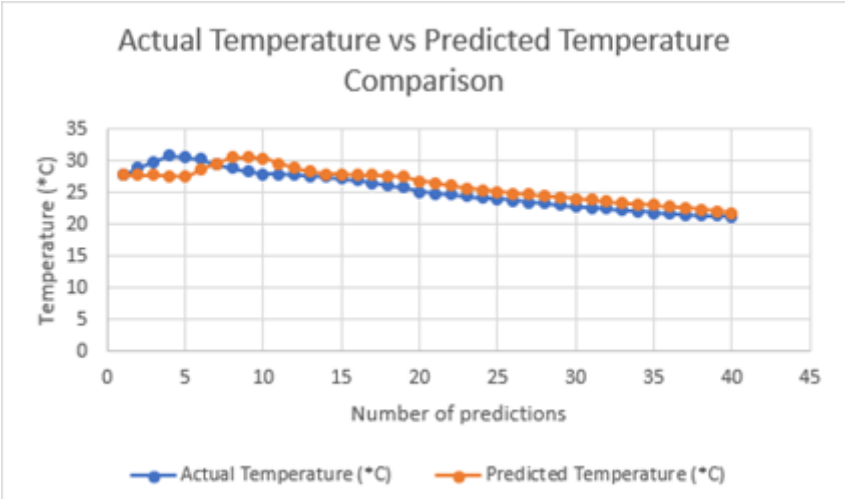

Figure 4.20: Actual Temperature vs Predicted Temperature Comparison

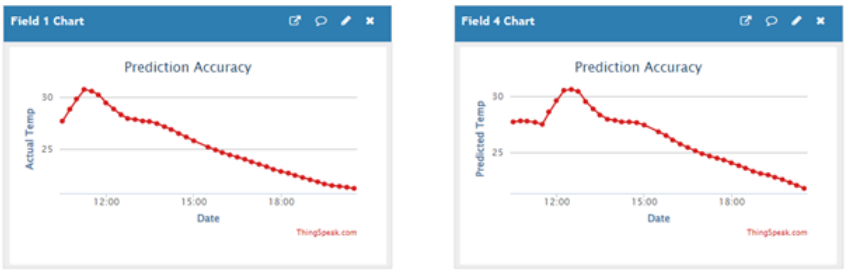

Figure 4.21: Actual Temperature vs Predicted Temperature Comparison (from ThingSpeak)

4.2.4 Accuracy vs Constraint Results

The ESP32 is constrained on storage, with only 448 KB. Table 4.1 below shows the reduction in size of the model during conversion to tflite format. This indicates that the conversion optimisations were successful and the expected lightweight version of the model. As indicated in the test results in the previous section on the TensorFlow Lite Model Performance Results, the accuracy of the predictions is not significantly affected by the optimisations during conversion.

Table 4.1: Comparison of model size before and after conversion to TensorFlow Lite

	MLP	CNN	LSTM
Model Size (MB)	0.017	0.181	1.448
TFLite Model Size (KB)	4.176	16.273	144.438

4.2.5 Execution Time Results

Figure 4.22 and Figure 4.23 show the execution time for running the inference on the ESP32 and Arduino Nano 33 BLE Sense respectively. Due to the 2-core 256 MHz processor of the ESP32 relative to the single-core 64 MHz processor of the Arduino, it was expected that the ESP32 would run the model and produce the temperature prediction faster.

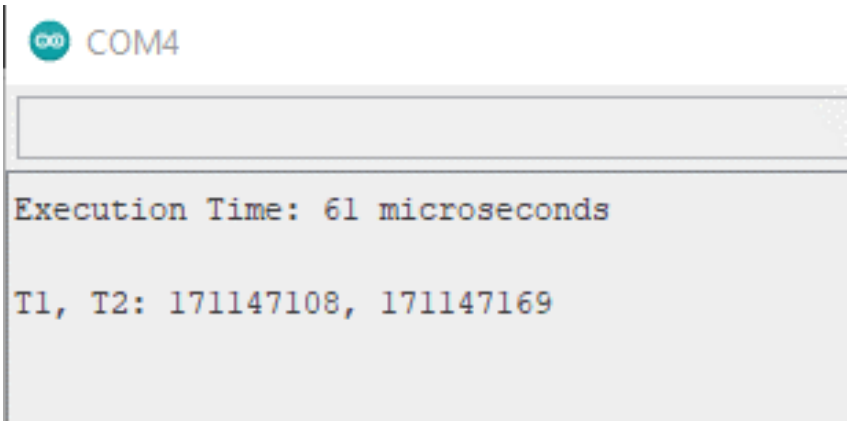

Figure 4.22: Inference execution time of ESP32

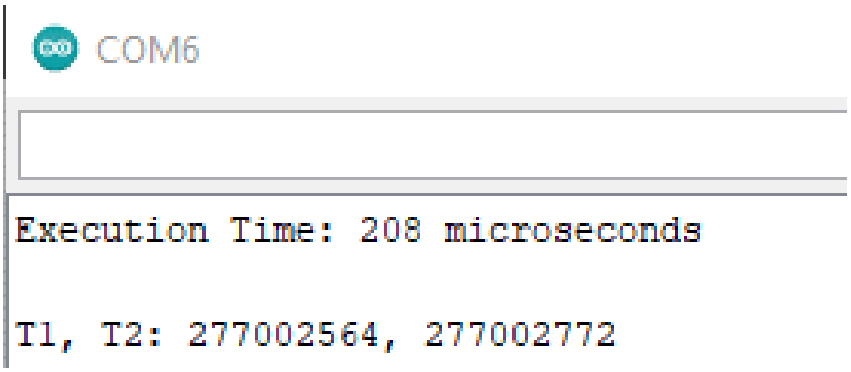

Figure 4.23: Inference execution time for Arduino Nano 33 BLE Sense

Chapter 5

Discussion

This chapter will give a summarised analysis of what the results presented in the previous chapter mean and how they relate to the literature presented in Chapter 2's literature review.

5.1 Model Performance

For the data without normalisation the LSTM model performed the best with RMSE and MAE values ranging from 0.12 - 0.4. The closer the RMSE and MAE values are to zero, the more accurate the predictions are. This was expected as RNN networks have the ability to keep the memory of an input within its layers making it more suitable for sequential time series applications. The results show that the LSTM model performs twice as much better than the other models. The MLP model was the least performing model, but its performance was not far off the CNN model with their RMSE and MAE values close together in the range of 0.25 - 1.6 for the MLP and 0.25 - 1.2 for the CNN.

For the normalised data, the CNN model performed the best with RMSE and MAE values ranging from 0.01 - 0.02. Normalising the data improved all models but its significantly improve the performance of the CNN model beyond the performance of the LSTM model. The number of data points used to train the model from the 1-hour and 10-minute data sets do not affect the performance of the CNN model. This is because the CNN model is able to identify patterns within the time steps of the data with its input shape (samples, timesteps, features) [28]. The CNN was expected to be one of the top performing models. Although it is mostly used in computer vision applications which use image matrices, time

series data input can be a 1D array whose information can be stored in the fully connected layers of the neural network [29].

The MLP was the least performing model in both cases. Unlike the other models, the MLP does not take the input as sequenced data therefore it will not clearly learn the data patterns as well as the CNN and LSTM. Although the MLP was the least performing model of the three, based on its the RMSE and MAE values, it produced accurate predictions close to the actual values in the validation and testing sets. This does not eliminate the model from being suitable for this application.

5.2 Target Device (Microcontroller)

When implementing TinyML applications, most literature mention the use of the Arduino Nano 33 BLE Sense. This is because of the many embedded sensors it has on its board making it compact and convenient for many applications. However, it does not connect to the internet which can be inconvenient for data collection. Although there are latency problems with uploading or extracting data in the cloud, this will not have any effect on the system itself. TinyML eliminates latency in inference since it occurs locally on the device. To easily analyse and evaluate the performance of the system, predicted and actual temperature values need to be collected to observe how well the system is operating. That data can be used to train another better model which will be deployed on the microcontroller. The ESP32 is able to connect to the internet making data collection easier.

The results show that the ESP32 runs the model faster with an execution time of 61 microseconds relative to the Arduino which takes 208 microseconds. These execution times are fast making both devices suitable for this real-time time series forecasting application. The 2-core 240 MHz CPU of the ESP32 compared to the single-core 64 MHz of the Arduino gives it the edge on performance.

5.3 Problems Faced

TinyML, also known as TensorFlow Lite for Microcontrollers is a fairly new implementation and is still under development by Google. The libraries used to run the TensorFlow Lite models on the microcontrollers are constantly being changed with new updates. There

is not sufficient documentation on most of the compatibility problems faced when trying to implement TinyML applications leaving open-source communities, like GitHub, to try to come up with alternative solutions to these problems. Complex models like this time series forecasting model are difficult to adapt for inference on microcontrollers. The trained LSTM and CNN models that were converted to TensorFlow Lite for inference could not be executed because of the complex models these neural networks create with operations that are not yet supported.

The other problem that was faced was trying to get historical Cape Town weather data to train the model. Most weather websites only offer this data at a price. Data was requested from Weather SA, but the request was not attended to. To solve this problem, the ESP32 and BME680 sensor was deployed outside for 24 hours to collect temperature, humidity, pressure, and air quality data which was uploaded to the cloud. This daily data was replicated to produce weather data equivalent to a seven-year period.

Chapter 6

Conclusions

This chapter will give an overview of the work done on the project. The overall system functionality will be reviewed to see if the project's deliverables and user requirements were met. Afterwards, a general conclusion on how the findings of this research is helpful to the topic in this field.

6.1 User Requirements Verification

6.1.1 Verification of UR001

The first requirement was to develop a Machine Learning model for time series forecasting. Three models were trained on Google Colab's cloud service using Python's TensorFlow library. Three different Deep Neural Networks, the Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), and the Long-Short Term Model, a Recurrent Neural Network were used to train the models. The root mean square error (RMSE) and mean absolute error (MAE) were the metrics used to verify the temperature prediction accuracy of these models, where the closer these values are to zero, the more accurate the predictions are. The CNN model was the best performing model for normalised data and the LSTM was the best performing model for unnormalised data.

6.1.2 Verification of UR002

After training the models with the three neural networks, they were converted to TensorFlow Lite which are lightweight versions of the original TensorFlow models. During this conversion the models were optimised for size and performance without risking much of the accuracy of the models. The size reductions of the models are displayed in Table 4.1.

6.1.3 Verification of UR003

Once the MLP, CNN, and LSTM models were converted to TensorFlow Lite (tflite), the tflite files were converted into a C-byte array to enable inference of the model on the ESP32 and Arduino Nano 33 BLE Sense using the Arduino IDE and its libraries. Due to the complexity of the CNN and LSTM models which contain operations which are not yet supported ny the TensorFlow Lite for Microcontrollers library, these models could not be executed on the microcontroller. However, the MLP model was able to be executed with success.

6.1.4 Verification of UR004

The prototype was deployed outside for 10 hours to test if the temperature predictions made were within the RMSE and MAE acceptable range of less than 3. The experiment was a success and the model was validated with the RMSE and MAE values being 1.45 and 1.3 respectively.

6.1.5 Verification of UR005

The hardware used to build and develop the system was below the R2,500 cost cap. All the software tools and cloud services were free. Table 6.1 shows the cost breakdown of the components used.

Table 6.1: Component Prices

Component	Price (R)	Source
Arduino Nano 33 BLE Sense	800	Diye Electronics
ESP32	270	Take a lot
BME680	345	Digi-Key

6.1.6 Verification of UR006

The execution time of the model on the ESP32 and Arduino Nano 33 BLE Sense was recorded and compared. The ESP32 and Arduino took 61 microseconds and 208 microseconds respectively to run the inference. Both execution times were fast thus achieving the real-time inference of the model using real-time temperature data from the BME680 temperature sensor. This will make the system regulate environmental conditions in its greenhouse application.

6.1.7 Verification of UR007

For user-friendliness, the SSD1306 OLED display was used to show the current environmental conditions. The system also displays any error that is encountered during the monitoring process. It also indicates some of the lengthy processes within the workflow such as the setup process and the loading of the initial temperature data used for the first inference.

6.2 General Conclusion

The project was a success with all the user requirements met and a working prototype that is able to monitor environmental conditions and forecast temperature one time step into the future. This research documentation can be used to redevelop this time series forecasting system or use the same technology for a similar application. This environmental monitoring and forecasting system is cheap and simple which can aid greenhouse automation systems to operate with minimal human supervision where crops can be effectively grown. Its ability to work offline and run the inference locally on the device can aid farmers who live in the rural areas were internet connection is poor and/or too expensive for them. The system can also make greenhouse farming sustainable by producing weather information that can help the farmers select a suitable crop to grow

suitable for such conditions.

Chapter 7

Recommendations

This section will give recommendations on what could have been done different in the design process/implementation of the system. It will touch on the difficulties faced during the research process and what could be done to overcome these difficulties. Research suggestions beyond the scope of this project will also be given to further develop the research topic.

7.1 Improvements in Executing this Project

7.1.1 Data Collection

To make the system more relevant to its target application in greenhouses, the environmental data used to train the models should have been collected in an actual greenhouse. Although external environmental conditions can have an effect on the internal environmental conditions in the greenhouse, collecting greenhouse data would make the trained model more reliable.

7.1.2 Backup Inference Data Transmission

Internet connection is not always reliable. Especially with the country's load shedding problems, connection can be lost anytime. To mitigate this problem, the Bluetooth Low Energy module of the ESP32 or the Arduino Nano 33 BLE Sense could be used to

transmit the actual temperature recorded by the sensors and the predicted temperature to a nearby device like a cellphone or another microcontroller with BLE capabilities that can be connected to a PC to collect the transmitted data.

7.1.3 Multistep Forecasts

The system could be developed to do multistep forecasts where it predicts the temperature more than a single time step into the future. This can used to observe the system's performance as the temperature of a specific time stamp will be predicted more than once. Observing how the system changes the prediction of that specific time stamp at different time steps would give more insight on how it adjusts to the changes in temperature conditions as well as how it uses the historical temperature to make its predictions.

7.2 Hardware Improvements

A PCB board designed for this specific application can be developed with all the required sensors embedded on it and pins available for easy plug-in of the SSD1306 OLED display.

7.3 Application Capabilities

This TinyML time series environmental monitoring and forecasting system is capable of being developed to use multivariate data, that is temperature, humidity, pressure and other environmental conditions to do a full multistep weather prediction. Temperature, precipitation, and even the outside condition, (whether its sunny, partly cloudy, or overcast), can be predicted. This enables a full weather forecast to be done offline without internet connection.

More research is needed to be done to enable inference of models trained by complex neural networks such as the CNN and LSTM. This improves the performance of the prediction model.

Use the IEEE numbered reference style for referencing your work as shown in your thesis guidelines. Please remember that the majority of your referenced work should be from journal articles, technical reports and books not online sources such as Wikipedia.

Bibliography

[1] O. O. Samson, M. O. Marvin, N. Jimmy and Z. Marco, "TinyML in Africa: Opportunities and Challenges," in IEEE, Madrid, 2021.

[2] S. Bharath, S. Simone, N. Duc-Duy, Y. Muhammad, W. Abdul, Y. Piyush, B. G. John and I. A. Muhammad, "TinyML Benchmark: Executing Fully Connected Neural Networks on Commodity Microcontrollers," in IEEE, New Orleans, 2.21.

[3] U. Nations, "UNDP," [Online]. Available: https://www.undp.org/sustainable-development-goalsindustry-innovation-and-infrastructure. [Accessed 4 November 2022].

[4] S. Ayad, A. Talli and S. L. Terrissa, "Toward agriculture 4.0: Smart farming environment based on robotic and IoT," in IEEE, Alkhobar, Saudi Arabia, 2021.

[5] D. Georgakopoulos, P. P. Jayaraman, M. Fazia, M. Villari and R. Ranjan, "Internet of Things and Edge Cloud Computing Roadmap for Manufacturing," IEEE, vol. 3, no. 4, pp. 66-73, 2016.

[6] "Statista," 22 August 2022. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/: :text=The%20number%20of%20Internet%20of,around%205%20billion%20consumer%2 [Accessed 5 October 2022].

[7] IBM, "IBM," [Online]. Available: https://www.ibm.com/za-en/cloud/what-is-edge-computing. [Accessed 6 October 2022].

[8] R. Mohanan, "Spice Works," 10 February 2022. [Online]. Available: https://www.spiceworks.com/tech/edge-computing/articles/what-is-edge-computing/. [Accessed 6 October 2022].

[9] M. Shafique, T. Theocharides, V. J. Reddy and B. Murmann, "TinyML: Current Progress, Research Challenges, and Future Roadmap," in IEEE, San Francisco, CA, USA, 2021.

[10] M. F. Alati, G. Fortino, J. Morales, J. M. Cecilia and P. Manzoni, "Time series analysis for temperature forecasting using TinyML," in IEEE, Las Vegas, 2022.

[11] B. Peter, C. Xuan, H. Eric and E. Chris, "Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware," in Research Gate, Waterloo, 2019.

[12] Tableu, "Tableu," [Online]. Available: https://www.tableau.com/learn/articles/time-series-analysis. [Accessed 11 October 2022].

[13] Australian Bureau of Statistics, "ABS," [Online]. Available: https://www.abs.gov.au/websitedbs/d3310114.nsf/home/time+series+analysis:+the+basics: :text= [Accessed 11 October 2022].

[14] Springer, "The Concise Encyclopedia of Statistics," Springer, New York, 2008.

[15] A. Alexandru-Toma and G. Ovidiu, "Unsupervised Machine Learning Algorithms Used in Deforested Areas Monitoring," in IEEE, Iasi, Romania, 2021.

[16] X. Zheng, L. Qinyi, Y. Run, G. Yifei and Y. Qian, "Image segmentation based on adaptive K-means algorithm," Springer Link, p. 68, 3 August 2018.

[17] G. Kalaivani and P. Mayilvahanan, "Air Quality Prediction and Monitoring using Machine Learning Algorithm based IoT sensor- A researcher's perspective," in IEEE, Coimbatre, India, 2021.

[18] J. Brownlee, Deep Learning for, 2018.

[19] V. LENDAVE, "AIM," 4 November 2021. [Online]. Available: https://analyticsindiamag.com/a-comprehensive-guide-to-representation-learning-for-beginners/. [Accessed 14 Octpber 2022].

[20] Tensorflow, "Tensorflow," 22 July 2022. [Online]. Available: https://www.tensorflow.org/lite/microcontrollers. [Accessed 14 October 2022].

[21] S.-I. Ramon and S. F. Antonio, "TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities," IEEE, vol. 20, no. 3, pp. 4-18, 2020.

[22] Arduino, "Arduino," 24 October 2022. [Online]. Available: https://store.arduino.cc/products/arduino-nano-33-ble-sense?selectedStore=eu. [Accessed 24 October 2022].

[23] Espressif, "Espressif," October 2022. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_-en.pdf. [Accessed 24 October 2022].

[24] A. o. Circuits, "Art of Circuits," [Online]. Available: https://artofcircuits.com/product/ssd1306-white-0-96-128x64-oled-display-i2c-interface. [Accessed 25 October 2022].

[25] B. Jason, T. Adrian and C. Zhe Ming, "Deep Learning with Python," in Develop Deep Learning Models with Tensorflow and Keras, Machine Learning Mastery, 2018.

[26] TensorFlow, "TensorFlow," Google, 3 August 2022. [Online]. Available: https://www.tensorflow.org/lite/guide/inference. [Accessed 1 November 2022].

[27] TensorFlow, "TensorFlow," Google, 28 October 2022. [Online]. Available: https://www.tensorflow.org/lite/guide/op_select_allowlist. [Accessed 1 November 2022].

[28] O. DIMITRE, "Kaggle," 2018. [Online]. Available: https://www.kaggle.com/code/dimitreoliveira/deep-learning-for-time-series-forecasting. [Accessed 1 November 2022].

[29] W. Aji Prasetya, U. Agung Bella Putra, E. Hakkun, P. Utomo, D. Felix Andika and H. Leonel, "Time-series analysis with smoothed Convolutional Neural Network," Springer Open: Journal of Big Data, no. 44, 2022.

Appendix A

Additional Files and Schematics

A.1 GitHub Link

To get access to all the project's code and files go to this Github repository

A.2 RMSE and MAE equations

$$RMSE = \sqrt{\frac{\sum (t - t')^2}{n}}$$

$$MAE = \frac{\sum |(t - t')|}{n}$$

where

$$t - actual\_temperature$$

$$t' - predicted\_temperaure$$

$$n - number\_of\_predictions$$

Appendix B

Addenda

B.1 Ethics Forms

Application for Approval of Ethics in Research (EIR) Projects
Faculty of Engineering and the Built Environment, University of Cape Town

ETHICS APPLICATION FORM

Please Note:
Any person planning to undertake research in the Faculty of Engineering and the Built Environment (EBE) at the University of Cape Town is required to complete this form **before** collecting or analysing data. The objective of submitting this application *prior* to embarking on research is to ensure that the highest ethical standards in research, conducted under the auspices of the EBE Faculty, are met. Please ensure that you have read, and understood the **EBE Ethics in Research Handbook** (available from the UCT EBE, Research Ethics website) prior to completing this application form: http://www.ebe.uct.ac.za/ebe/research/ethics1

APPLICANT'S DETAILS		
Name of principal researcher, student or external applicant		Takura Tyrone Kawome
Department		Electrical Engineering
Preferred email address of applicant:		kwmtak001@myuct.ac.za
If Student	Your Degree: e.g., MSc, PhD, etc.	BSc/Eng
	Credit Value of Research: e.g., 60/120/180/360 etc.	40
	Name of Supervisor (if supervised):	Dr. Joyce Mwangama
If this is a researchcontract, indicate the source of funding/sponsorship		
Project Title		Time Series Analysis for Environmental Monitoring and Forecasting Using TinyML

I hereby undertake to carry out my research in such a way that:

- there is no apparent legal objection to the nature or the method of research; and
- the research will not compromise staff or students or the other responsibilities of the University;
- the stated objective will be achieved, and the findings will have a high degree of validity;
- limitations and alternative interpretations will be considered;
- the findings could be subject to peer review and publicly available; and
- I will comply with the conventions of copyright and avoid any practice that would constitute plagiarism.

APPLICATION BY	Full name	Signature	Date
Principal Researcher/ Student/External applicant	Takura Tyrone Kawome	ttk	19/08/2022
SUPPORTED BY	Full name	Signature	Date
Supervisor (where applicable)	Joyce Mwangama		19/08/2022

APPROVED BY	Full name	Signature	Date
HOD (or delegated nominee) Final authority for all applicants who have answered NO to all questions in Section 1; and for all Undergraduate research (Including Honours).			
Chair: Faculty EIR Committee For applicants other than undergraduate students who have answered YES to any of the questions in Section 1.			

Figure B.1: Ethics Form