Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Incorporating Machine Learning in a Microcontroller-Driven Sensor System for Monitoring Cold Chain Pharmaceutical Products

Aaron Darcy

BSc (Hons) in Computing 2023/2024

Incorporating Machine Learning in a Micro-Controller Driven Sensor System for Monitoring Cold Chain Pharmaceutical Products

Author: Aaron Darcy

Supervised by: Paul Moran

A thesis submitted in partial fulfilment of the requirements for “Research in Computing with Emerging Technologies” and “Project Development”

BSc (Hons) in Computing

Submitted to Atlantic Technological University
Arna chur isteach chuig Ollscoil Teicneolaiochta an Atlantaigh May 2023

Declaration

I hereby certify that the material, which I now submit for assessment on the programmes of study leading to the award of Bachelor of Science in Computing, is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution. I understand that it is my responsibility to ensure that I have adhered to ATU's rules and regulations.

I hereby certify that the material on which I have relied on for the purpose of my assessment is not deemed as personal data under the GDPR Regulations. Personal data is any data from living people that can be identified. Any personal data used for the purpose of my assessment has been pseudonymised and the data set and identifiers are not held by ATU. Alternatively, personal data has been anonymised in line with the Data Protection Commissioners Guidelines on Anonymisation.

I consent that my work will be held for the purposes of education assistance to future students and will be shared on the ATU Donegal (Computing) website (www.lyitcomputing.com) and Research THEA website (https://research.thea.ie/). I understand that documents once uploaded onto the website can be viewed throughout the world and not just in the Ireland. Consent can be withdrawn for the publishing of material online by emailing Thomas Dowling; Head of Department at thomas.dowling@lyit.ie to remove items from the ATU Donegal Computing website and by email emailing Denise McCaul; Systems Librarian at denise.mccaul@lyit.ie to remove items from the Research THEA website. Material will continue to appear in printed formats once published and as websites are public medium, ATU cannot guarantee that the material has not been saved or downloaded.

Signature of Candidate Date

Acknowledgements

First and foremost, I would like to thank my family, for their support and encouragement throughout my academic journey.

I am profoundly grateful to my supervisors, Kevin Meehan, and Paul Moran, for their guidance and expertise. Kevin Meehan, my supervisor during the first semester, provided the initial direction and insightful feedback that shaped the early stages of my research. Paul Moran, who supervised me in the second semester, further guided and refined my research with his expertise, for which I am very thankful.

I would also like to express my gratitude to all the lecturers at ATU Galway and ATU Donegal. Their dedication and commitment to teaching have greatly enriched my learning experience and inspired me throughout my academic journey.

A special thanks to my employer Kelsius, for allowing me access to the historical sensor data which formed the foundation of my research. This gesture was greatly appreciated & crucial in the practical application of the machine learning model. Particularly Damien McKeever, James Boyce, and Freddy Lusson for their insightful contributions and willingness to share their expertise, which have been significantly beneficial to my research. Additionally, I would like to thank Daniel Fecowicz for his efforts in preparing the data.

Abstract

The pharmaceutical industry heavily relies on the cold chain to maintain the integrity of temperature-sensitive products during storage and transportation. Effective monitoring of these conditions is paramount to ensuring product quality and safety. This thesis explores the incorporation of Machine Learning techniques within a microcontroller-driven sensor system designed to enhance the monitoring of cold chain environments for pharmaceutical products. The system leverages a combination of an ESP32 microcontroller sensor combination to collect real-time temperature data, which is then processed and analysed using Long Short-Term Memory (LSTM) models to predict potential deviations before they occur.

The research was conducted in two phases. Initially, the system's capability to collect and transmit sensor data accurately was established, followed by the implementation and validation of LSTM models to predict temperature fluctuations. The models were trained on historical data provided by Kelsius, allowing for the assessment of their predictive accuracy and reliability.

The findings confirm that integrating LSTM models into the sensor system significantly improves the prediction of temperature anomalies, thus offering a proactive approach to cold chain management. The best-performing model predicted temperature deviations with over 96% accuracy, demonstrating the feasibility and effectiveness of ML in real-world applications within the pharmaceutical cold chain.

This study not only validates the integration of ML models into existing microcontroller-based monitoring systems but also illustrates the potential enhancements in predictive accuracy and operational efficiency, providing a robust framework for future advancements in the field.

Acronyms

Acronym	Definition	Page
EFPIA	European Federation of Pharmaceutical Industries and Associations	1
IoT	Internet Of Things	1
RF	Radio Frequency	3
SSL	Secure Socket Layer	33
TLS	Transport Layer Security	33
ARIMA	Auto Regressive Integrated Moving Average	5
ML	Machine Learning	6
AI	Artificial Intelligence	8
TSA	Time Series Analysis	8
RL	Reinforcement learning	8
RNN	Recurrent Neural Networks	9
ANN	Artificial Neural Networks	10
CNN	Convolutional Neural Networks	10
LSTM	Long-Short Term Memory	11
CPU	Central Processing Unit	12
GPU	Graphics Processing Unit	12
DSP	Digital Signal Processor	12
ASIC	Application-Specific Integrated Circuit	12
FPGA	Field Programmable Gate Array	12
FDA	Food & Drug Administration	15
ETA	Estimated Time of Arrival	16
LCD	Liquid Crystal Display	19
IDE	Integrated Development Environment	19
Co2	Carbon Dioxide	22
LED	Light Emitting Diode	23
NFR	Non-Functional Requirements	34
SD	Secure Digital	36
UI	User Interface	46
NaN	Not a Number	61

Table of Contents

Table of Contents

Table of Figures

Table of Tables

Table of Code Listings

Introduction

In the world of healthcare and pharmaceuticals, the stakes are notably high both economically and ethically. This industry bridges vast economic investments with the profound responsibility of shaping health outcomes. Decisions are not just about profits, but also impact the well-being of millions. Such significance is evident when considering the financial commitments.

According to a report conducted by the European Federation of Pharmaceutical Industries and Associations (EFPIA 2022), the research-based pharmaceutical industry in Europe invested an estimated €41,500,000 in Research & Development. Such significant investments reflect not just the immense market value, which stands at an estimated €236,090,000 in Europe alone (The Pharmaceutical Industry in Figures 2022), but also the profound commitment to advancing patient care and health outcomes. A cornerstone of this commitment is ensuring the safe and controlled storage of cold chain pharmaceutical products. These products, often pivotal to patient health and even survival, must be stored under stringent conditions to retain their efficacy ('Control and Monitoring of Storage and Transportation Temperature Conditions for Medicinal Products and Active Substances' 2020). Thus, while the economic implications of proper storage are evident, the human implications are also crucial. Ensuring the integrity of these medicines is not just about safeguarding economic assets but, more importantly, about preserving and enhancing the quality of patient life.

Temperature deviations, even slight ones, can adversely affect the efficacy and safety of drugs and vaccines. Traditional vaccine fridges have integrated monitoring systems, they rely heavily on predefined thresholds and manual checks, and while effective, may not be agile enough to predict and prevent potential temperature anomalies (HSE 2020, p.5). With the proliferation of Internet of things (IoT) and capabilities on microcontrollers, there arises an opportunity to enhance these monitoring systems. The integration of machine learning into microcontroller-driven sensor systems enhances their capabilities in predictive analysis and real-time adjustments, which is crucial in mitigating the effects of false positives and anomalies. Such false positives can be both costly and disruptive, especially in critical applications like temperature monitoring. Machine learning algorithms help in accurately distinguishing between genuine issues and false alarms, thereby reducing unnecessary interventions, and ensuring the reliability of the system. This is particularly valuable in environments where precision and timely response are crucial.

1.1. Purpose

The purpose of this thesis is to investigate the integration of machine learning techniques into a microcontroller-driven sensor system designed for the purpose of enhancing the monitoring and management of cold chain pharmaceutical products. This research aims to explore the use of machine learning algorithms to improve the accuracy and reliability of temperature-sensitive pharmaceutical product tracking within the industry, ultimately contributing to the preservation of product integrity and patient safety.

1.2. Background

Conventional refrigerators typically use built-in sensors and fixed rules to trigger alarms for temperature changes. These systems generally operate in isolation and lack adaptability to dynamic conditions or the capability to foresee risks using both historical and current data. The drawbacks of such limitations can lead to significant consequences, including the spoilage of products, financial losses, and, more critically, the potential harm to patients if affected drugs and vaccines are used (Gebremariam et al. 2019).

Integrating machine learning into a microcontroller-driven sensor presents a solution for enhancing cold chain monitoring. Machine Learning algorithms can offer adaptable monitoring by examining patterns predicting temperature deviations and providing proactive measures. Moreover microcontrollers, with their size and low power consumption make it practical to implement Machine learning based solutions without increasing the size or power requirements of the monitoring system.

1.3. Research Question

How can machine learning algorithms be incorporated effectively into a microcontroller-driven sensor system to enhance the accuracy and predictability of monitoring cold chain pharmaceutical products, and what improvements in detection and prediction of temperature excursions can be expected compared to traditional monitoring systems and fridges.

1.4. Aims and Objectives

The aim of this dissertation is to research and test relevant Machine Learning Models and incorporate it with microcontroller-driven sensor system. The system as a whole will enable the safeguarding and monitoring of cold chain pharmaceutical products. To reach this standard the following objectives must be met.

- Analyse the shortcomings/gaps of current temperature monitoring systems in drug fridges.
- Design & develop a micro controller that collects and sends data from various types of wireless sensors.
- Design & build a simple Web based solution for users to view and run reports on temperature data.
- Compare Radio Frequency (RF) and Wi-Fi based sensors to see which one is more appropriate for the system.
- Implement machine learning models to assist with Anomaly Detection and Time-Series Forecasting.
- Research & experiment with different models with test data to ensure the appropriate models are in place.
- Curate a training dataset with diverse temperature readings from drug fridges for machine learning model efficacy in Anomaly Detection and Forecasting.
- Implement and compare secure communications methods like Secure Socket Layer (SSL) /Transport Layer Security (TLS) to ensure data is encrypted in transit.
- Implement the new WPA3 Wi-Fi protocol to ensure secure wireless communication between the micro-controller, sensors, and the network.

1.5. Report Outline

The thesis begins with an Introduction that sets the stage by establishing the research's purpose, context, and objectives, while also outlining the overall structure of the work. This is followed by a Literature Review which dives deep into machine learning techniques, time series analysis, signal processing, and the challenges with cold chain monitoring within the realm of pharmaceuticals. This section also includes a detailed comparative analysis of various micro-controllers such as ESP32, Arduino Uno R3, and Raspberry Pi Pico 2040. The Design and Methodology section transitions into the practical aspects of the thesis, where both system and application requirements are described. It details the proposed system architecture and illustrates the design through various diagrams and storyboards. Next, the Implementation section elaborates on the development and setup processes, the integration of machine learning models with a microcontroller and sensor, the configuration of the web application component, and the data handling techniques utilized throughout the project. The thesis then progresses to the testing strategy section, where findings from the implementation and testing phases are presented, analysing the performance of the machine learning models and the efficacy of the micro-controller systems and web application in real-world applications. The Conclusion summarizes the key findings, discusses the limitations of the current study, and offers suggestions for future research.

2. Literature Review

2.1. Introduction

This literature review aims to dissect the intersection of machine learning, microcontrollers, embedded systems, and specialized sensors within the realm of cold chain monitoring in pharmaceuticals.

Machine Learning:

This section delves into the fundamentals and application of machine learning algorithms in data analysis and predictive modelling.

Time Series Analysis:

This segment explores the use of time series analysis in cold chain monitoring, including methodologies such as Auto Regressive Integrated Moving Average (ARIMA), Recurrent Neural Networks (RNN), and Long Short-Term Memory (LSTM). It discusses how these techniques help in forecasting and analysing data over time.

Data Science and Signal Processing:

Explores the role of data science in analysing different signals. It discusses concepts of signal preprocessing and filtering, underscoring how these techniques contribute to accurate data analysis and decision-making in pharmaceutical monitoring.

Cold Chain Monitoring in Pharmaceuticals:

This segment focuses on the criticality of maintaining conditions in the pharmaceutical supply chain. It reviews current practices, challenges, technological advancements, and the impact of temperature excursions on pharmaceutical products, highlighting the necessity of advanced monitoring solutions.

Microcontrollers & Embedded Systems:

In this section, the focus is on technological advancements of microcontrollers and embedded systems, highlighting how these systems have evolved to efficiently process data. This chapter also contains a comparative analysis of three different micro-controllers.

Security in Embedded Systems:

This section addresses the security aspects of embedded systems. It covers the vulnerabilities, threats, and strategies in securing embedded systems.

Sensors in Cold Chain Pharmaceutical Monitoring Systems:

This segment scrutinizes the diverse sensor types used in pharmaceutical product monitoring, focusing on regulatory compliance and adherence to standards. Additionally, the section includes a comparative analysis of RF and IoT sensors.

2.2. Machine Learning

As illustrated in figure 1 below Machine learning (ML) is a specialized subset of Artificial Intelligence (AI), a field that has become an indispensable tool in our daily lives. AI manifests in various forms, from the smartphone assistants we interact with daily, to the personalized recommendations we receive on streaming services. Its influence is both subtle and profound, seamlessly integrating into our routines. It's important to note that while all machine learning is a form of AI, not all AI is machine learning (Microsoft 2023). AI encompasses a broader spectrum of technologies, including rule-based systems and logic programming, which don't necessarily rely on learning from data. Machine learning specifically focuses on creating systems that learn and improve their performance (or intelligence) based on the data they process. This is achieved through its unique ability to automatically adjust algorithms based on experience, a crucial aspect for handling and making predictions from complex data sets.

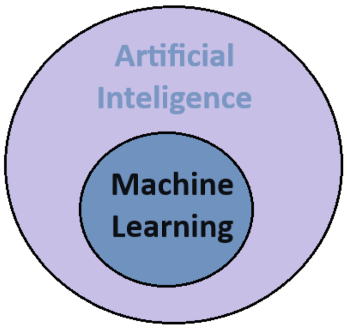


Figure 1 – ML is a subset of AI.

Machine learning comprises various approaches and methodologies, each suited to specific types of problems and data. Machine learning models can be classified into three primary categories: supervised learning, unsupervised learning, and reinforcement learning. These categories differ in how they learn from data and make predictions or decisions, representing the diverse capabilities and applications of machine learning. This section goes into detail on each to determine which suitable for the project (IBM 2023).

2.2.1. Supervised Learning

Supervised learning is a method where a model is trained using a dataset that includes both the input and the corresponding correct output. This approach, which involves matching each piece of training data with the correct answer, is ideal for tasks like regression and classification. According to IBM (2023), the training process of supervised learning is a complex one, requiring careful tuning of the weights in the algorithms, which is a critical part of the cross-validation process. Most supervised learning algorithms, especially neural networks, involve weights. These weights are parameters within the model that are adjusted during the training process. Initially, these weights are set randomly, and as the training progresses, they are incrementally adjusted. The adaptability of supervised learning is evident in its widespread application in real-world scenarios, handling tasks that range from classifying spam to conducting predictive analysis like time series forecasting.

2.2.2. Unsupervised Learning

Unsupervised learning algorithms distinguish themselves from supervised learning by handling unlabelled data. These models are designed to delve into the data's inherent structure, identifying patterns, and correlations. These models are also suitable for clustering as seen in figure 2. As Google Cloud (2023) notes, unsupervised learning is particularly adept at processing large volumes of unlabelled data, greatly simplifying how businesses and systems derive insights from such datasets. This approach autonomously uncovers data structures, detecting patterns and relationships without the need for manual, human-led instruction. This capability renders unsupervised learning models particularly effective in discerning patterns and discrepancies within unstructured data (Google Cloud 2023; Data Camp 2023).

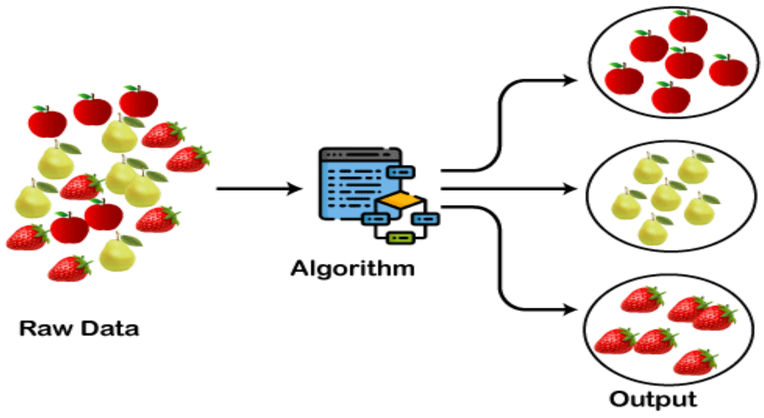


Figure 2 – Clustering In Machine Learning Source:('Clustering in Machine Learning: 5 Essential Clustering Algorithms' 2023)

2.2.3. Reinforced Learning

Reinforcement learning (RL) is a distinctive paradigm within machine learning where an agent learns from interacting with its environment. This process involves the RL agent making decisions and learning from the outcomes of its actions. It operates on a system of rewards and penalties, where the agent is incentivized for successful actions and learns to refine its strategy for optimal decision-making (Data Tonic 2023). The strength of RL lies in its ability to solve problems that require a series of sequential decisions, making it versatile across various domains (Li 2018, p.5). This approach is particularly effective in situations where direct instruction is not feasible, and learning through experimentation is key.

2.3. Time Series Analysis

After exploring the primary categories of machine learning and their characteristics, it becomes apparent how these methodologies can be applied to specific domains, such as Time Series Analysis (TSA).

Time series analysis involves examining a series of data points that are gathered at regular intervals over a certain time frame. Unlike sporadic or random data collection, time series analysis focuses on consistent and periodic recording of data. More than just gathering data over time, this analysis is distinct because it reveals how variables evolve and change as time progresses (Tableau 2023). This segment of data analysis leverages both supervised and unsupervised learning techniques to forecast future values based on historical data and to uncover patterns or anomalies in time-ordered data sets (Gupta 2022).

Time Series Analysis is particularly relevant in domains like finance, temperature forecasting, and inventory management, where understanding trends, variances, and potential outliers is vital (Pandian 2021).

The application of machine learning in Time Series Analysis varies based on the nature of the time series data and the specific objectives of the analysis (National Institute of Standards and Technology 2023). For instance, in supervised learning, models like ARIMA and Recurrent Neural Networks (RNNs) are employed to predict future data points in a series, whereas unsupervised learning methods are used to detect anomalies or identify underlying structures in the data without predefined labels. The choice of method depends on whether the focus is on forecasting future trends or exploring data for hidden patterns and irregularities.

2.3.1. Auto Regressive Integrated Moving Average (ARIMA)

The ARIMA model is recognized as one of the most popular and extensively utilized statistical approaches for forecasting in time-series analysis (Nathaniel 2021).

The ARIMA model represents an extension of the simpler Autoregressive Moving Average (ARMA) model. Both models are employed for predicting future data points in time series. ARIMA functions as a type of regression analysis, illustrating the relationship of a dependent variable with other variables that are subject to change. This model primarily aims to forecast future movements in a time series by analysing the differences between sequential values, rather than the absolute values themselves. ARIMA is particularly useful in situations where the data exhibits signs of non-stationarity. In the context of time series analysis, transforming non-stationary data into stationary data is a standard practice (Chao 2021).

When breaking down the ARIMA model it can be broken down into the following three components (Analytics Vidhya 2023):

Auto Regressive (AR): The Autoregressive model, which is represents a kind of random process. In this model, the output depends linearly on its own previous values, which means it uses a certain number of previous lagged data points, or past observations, to predict the next value.

Moving Average (MA): This refers to the Moving Average model, where the output is linearly dependent on the current and various past instances of random variation.

Integrated(I): Integrated in this context means the process of making the time series data stationary by differencing, essentially removing the seasonal and trend components from the data. This can be done by subtracting the current value from the previous value, to stabilize the mean of the time series. This process might be repeated multiple times, depending on the degree of differencing needed to achieve stationarity. Tuning of these parameters to find the optimal value is crucial. The right level of differencing ensures stationarity in the time series. This step is essential for the ARIMA model's accuracy in forecasting and identifying data trends. The ARIMA model is commonly represented as ARIMA (p, d, q), where the parameters p, d, and q are defined as (Nathaniel 2021):

p: The number of lags, or time delays, used in the Autoregressive model, AR(p).

d: The degree of differencing, or the number of times the data has been differenced, which means subtracting past values from current values, I(d).

q: The order of the Moving Average model, MA(q), indicating the number of lagged forecast errors in the prediction equation.

In conclusion, the ARIMA model is an essential tool for forecasting in time series analysis, ideal for interpreting and predicting trends, especially in non-stationary data.

2.3.2. Recurrent Neural Networks (RNNs)

A neural network is a type of machine learning model structured to emulate the human brain's functionality and architecture. These networks consist of complex interconnections between nodes, also known as neurons, that work together to solve intricate challenges. Often called artificial neural networks (ANNs) or deep neural networks, they fall under the umbrella of deep learning (Yasar 2023).

RNNs are a specialized form of artificial neural networks crafted for handling sequential data. They excel in tasks that involve sequences, such as dealing with time series data, processing voice and natural language, and similar sequential tasks (Biswal 2023). Training a RNN bears resemblance to training a conventional neural network (CNN). The backpropagation algorithm is employed here as well, albeit with a slight modification. In an RNN, since all-time steps share the same parameters, the gradient at each output is influenced not just by the computations of the current time step, but also by those from preceding time steps (Chauhan 2020). See below in figure 3 of a RNN Structure which visualizes each step.

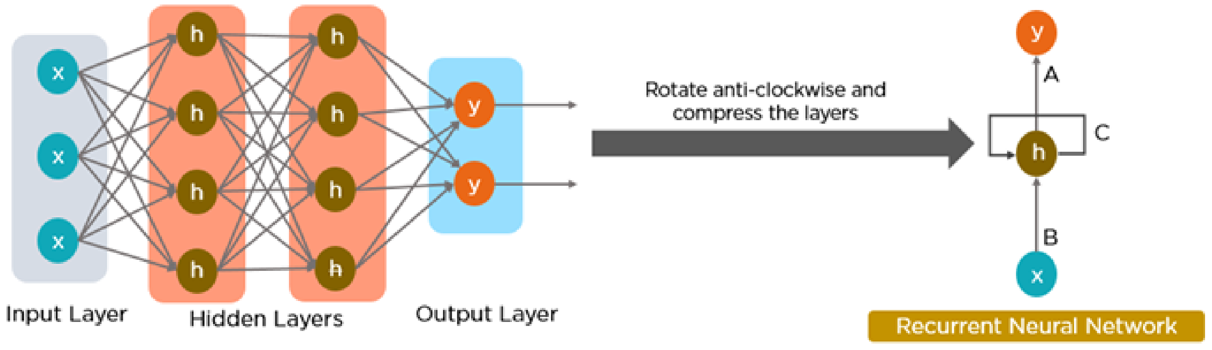


Figure 3 – RNN Structure (Kalita 2022)

2.3.3. Long Short-Term Memory (LSTM)

Conventional RNNs, while robust in many aspects, often grapple with the vanishing gradient problem (Whitfield 2023). This issue arises when the gradients (which is for guiding the adjustment of weights during training to minimize error and optimize the model's performance) used in training the network become increasingly smaller, hindering the network's ability to learn from and remember long-term patterns in sequential data (Kalita 2022).

LSTM networks address this limitation effectively. LSTMs incorporate specialized memory cells and a system of gates, ensuring a more consistent and controlled flow of gradients even across extended sequences. This architecture markedly enhances the network's proficiency in capturing long-term dependencies (Buhl 2023).

The LSTM model includes three gates, each employing sigmoid functions that output values between 0 and 1, signifying how much of each component should be allowed to pass through. These gates are (Saxena 2021):

Input Gate: It manages the entry of new data into the memory cell, selectively retaining crucial information for the task at hand.

Forget Gate: This gate decides which information to remove from the memory cell, allowing the LSTM to discard irrelevant data.

Output Gate: It controls the flow of information from the LSTM to the output, determining which parts of the current cell state are used to generate the output.

Each of these gates is trained through backpropagation to react not just to the current input but also to the information from the previous hidden state. Backpropagation is a fundamental training algorithm used in neural networks, where the network's errors are propagated backwards, starting from the output layer to the input layer. This process involves adjusting the weights of the network in a way that minimizes these errors, thereby enhancing the network's performance and accuracy. This training method enables the LSTM to selectively retain or discard information, a critical feature for effectively capturing long-term dependencies in sequential data processing. (Brownlee 2017).

In conclusion, Recurrent Neural Networks, especially LSTMs, are crucial in deep learning for sequential tasks, overcoming the vanishing gradient problem with their unique architecture. Their proficiency in processing long-term dependencies makes them vital for complex applications like time series analysis and language processing.

2.4. Signal Processing

Signal processing, a fundamental aspect of modern computing and electronics, involves the analysis, manipulation, and interpretation of signals, playing a crucial role in converting real-world information like sound, images, and environmental data into digital forms for processing (Harvie 2023).

In the past, signal processing was exclusively conducted in the analogue realm, utilizing individual components such as resistors, capacitors, and inductors to create signal filters. While this practice continues, the latter part of the 20th century saw a significant shift towards the digitization of various types of data. Consequently, signal processing has largely moved from analogue to digital methods. Currently, digital signal processing predominantly occurs through software. This software can operate on a desktop computer's Central Processing Unit (CPU) or Graphics Processing Unit (GPU), or within a smart device. For tasks that require more processing power, it is executed on specialized hardware like Digital Signal Processors (DSPs), Application-Specific Integrated Circuits (ASICs), or Field Programmable Gate Arrays (FPGAs), as well as on advanced computer mainframes (Smith 2023).

2.4.1. Data Science & Signal Processing

According to Amberle McKee (2023), Signal processing is a core aspect of data science, focusing on the extraction, analysis, and modification of signals and time-series data. It encompasses a wide and sometimes intricate area of study.

Signal analysis involves examining signal attributes like shape, amplitude, frequency, phase, energy, entropy, correlation, independence, stationarity, and causality. This helps in understanding signals' meanings and identifying their origins. On the other hand, signal processing focuses on altering signals to achieve specific objectives, including noise reduction, data compression, encoding, transmission, filtering, source separation, detection, and classification. This enhances signal quality, efficiency, robustness, and reliability, and aids in extracting significant information concealed within the signals (Automata 2023).

2.4.2. Preprocessing and Filtering Concepts

Data preprocessing, a crucial phase of data preparation, involves transforming raw data for subsequent processing or analysis. Traditionally vital for data mining, these techniques are now also essential for training and running inferences on machine learning and AI models. By converting data into a more manageable format, data preprocessing facilitates more efficient processing in data mining, machine learning, and other data science activities, ensuring accuracy from the earliest stages of the AI and machine learning development pipeline (Lawton 2023).

2.4.2.1 Missing Data Points

Data gaps can occur due to various reasons, such as intentional non-responses in surveys, malfunctions in electrical sensors, or other causes. When this occurs, there's a risk of losing important information (Ogunbiyi 2022).

Resampling: Resampling is a method employed to normalize the intervals in a dataset. This technique may include up sampling, which increases the frequency of data points, or down sampling, which reduces the frequency, to achieve a consistent time-series (McKee 2023).

Interpolation: When dealing with missing or estimable data points, interpolation techniques are utilized. Popular methods of interpolation encompass linear interpolation, spline interpolation, and interpolation based on time (McKee 2023).

2.3.2.2 Noisy Data

Errors in datasets can arise from human mistakes during data collection or inaccuracies in the instruments used for gathering data, leading to what is known as noise. If not properly addressed, this noise can pose problems in machine learning algorithms. Without adequate training to distinguish between actual patterns and noise, the algorithm may mistakenly interpret this noise as a significant pattern, leading to incorrect generalizations (Skan Editorial Staff 2023).

Noise Filters: Low-pass, high-pass, and band-pass filters are techniques in frequency-based filtering that selectively allow or block certain frequencies in a signal. Low-pass filters let through low-frequency signals and attenuate higher frequencies, useful for noise reduction and preserving slow trends. In contrast, high-pass filters enable the passage of higher frequencies while eliminating lower ones, ideal for accentuating brief events or rapid changes in a time series. Band-pass filters, on the other hand, permit only a specific range of frequencies to pass, blocking frequencies outside this range (McKee 2023).

Moving Average Filter: A moving average filter is a fundamental method for eliminating noise from a signal, acting as a basic type of low-pass filter. When a signal is processed through this filter, it eliminates higher frequency components from the result. Unlike a conventional low-pass filter, which is adept at isolating a specific signal frequency, the moving average filter takes a more straightforward route to simply smooth the signal. The concept behind it is straightforward: the filter computes the average of the most recent "M" data points in the signal, using this average as the output (Garberman 2020).

2.5. Cold Chain Monitoring in Pharmaceuticals

Cold chain monitoring in pharmaceuticals refers to a temperature-controlled supply chain used to maintain and distribute products in a temperature-specific range (WHO India n.d.). It involves a series of storage and distribution activities that keep pharmaceutical products, such as vaccines, biologics, and certain medications, within a prescribed temperature range from the point of manufacture to the point of use. Given its critical role in ensuring the safety and potency of temperature-sensitive products, research has consistently highlighted the importance of stringent cold chain management for pharmaceuticals. Studies reveal that even minor deviations in temperature can compromise the efficacy of drugs, leading to significant health and economic consequences (Feyisa *et al.* 2022; Airfinity 2023).

2.5.1. Challenges in Cold Chain Monitoring

One of the most significant challenges in cold chain monitoring, as identified in the literature, is maintaining consistent temperature control across different stages of the supply chain as

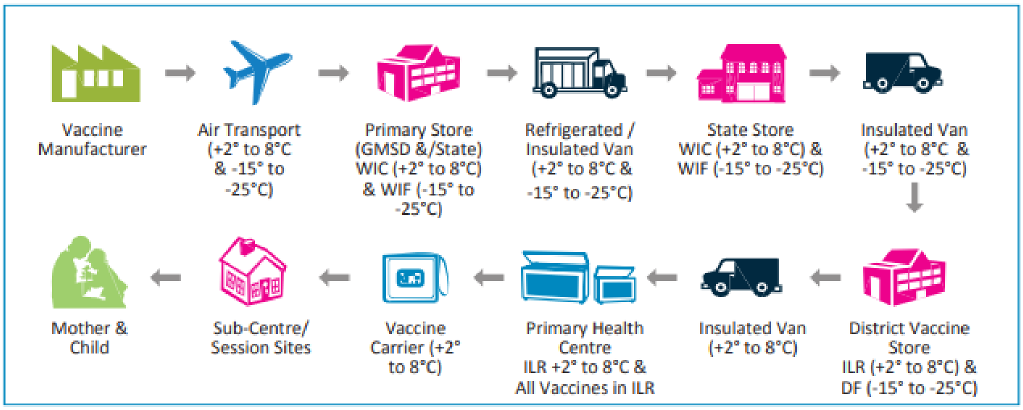

Vaccine Manufacturer
Air Transport (+2° to 8°C & -15° to -25°C)
Primary Store (GMSD &/State) WIC (+2° to 8°C) & WIF (-15° to -25°C)
Refrigerated / Insulated Van (+2° to 8°C & -15° to -25°C)
State Store WIC (+2° to 8°C) & WIF (-15° to -25°C)
Insulated Van (+2° to 8°C & -15° to -25°C)
District Vaccine Store ILR (+2° to 8°C) & DF (-15° to -25°C)
Insulated Van (+2° to 8°C)
Primary Health Centre ILR +2° to 8°C & All Vaccines in ILR
Vaccine Carrier (+2° to 8°C)
Sub-Centre/ Session Sites
Mother & Child

Figure 4 - Stages Temperature-Controlled Supply Chain Within Pharmaceuticals (WHO India n.d.)

displayed in Figure 4 above. The complexity is compounded by the involvement of multiple environments and varying logistical scenarios, each introducing potential risks for temperature excursions. Additionally, evolving regulatory requirements across jurisdictions like the United States (U.S), European Union (EU), and Canada add complexity to implementing a universally compliant cold chain monitoring system. The U.S. focuses on detailed Food and Drug Administration (FDA) regulations for drug storage and handling (U.S FDA 2023; 21 CFR 205.50, 21 CFR 203.32). These regulations are available to view on the U.S federal register. Canada and the EU similarly emphasizes temperature control and national vaccine storage guidelines to ensure quality and safety in pharmaceutical production (EUR – LEX 2023; Federal laws of Canada 2023). These diverse regulations underline the challenge of aligning cold chain management practices across different legal and operational frameworks.

The assurance of medicine quality, especially in low and middle-income countries, is also a pressing issue. Caroline Brogan (2020) highlights that these regions often grapple with challenges stemming from weak regulatory oversight, limited resources, and a lack of infrastructure. This situation poses a significant risk to the efficacy and safety of pharmaceuticals in these regions.

The severity and frequency of temperature deviations in the cold chain are further emphasized by a 2019 survey conducted by Peli Biothermal. Their findings reveal a concerning trend:

“Awareness of temperature-control requirements is higher than ever, yet our survey indicates that temperature excursions continue to occur frequently. Almost half (44.6%) of the survey respondents report experiencing multiple temperature excursions annually, and 16% encounter such issues on a monthly basis. What’s more alarming is the extent of these deviations; 41% of these excursions exceed a four-degree variance, and 21% surpass an eight-degree variance.”

The findings from the Peli Biothermal survey in 2019, highlighting the frequent and significant temperature excursions in the cold chain, set a compelling context for the urgency and necessity of advancing cold chain monitoring technologies. The alarming frequency and severity of these temperature deviations underscore the need for more robust, reliable, and intelligent monitoring solutions, especially in the realm of pharmaceutical logistics where product integrity is non-negotiable.

2.5.2. Technological Advancements and Solutions

A recurring theme in recent literature is the potential of integrating advanced technologies into the cold chain monitoring process. Internet of Things (IoT)-based sensor networks are gaining traction for their ability to provide real-time data and wider coverage in transit to their destination (Inc 2022). These systems can provide temperature reports, estimated time of arrivals (ETAs), and historical temperature records, which are essential for managing temperature-controlled products on the go efficiently.

Emerging research also focuses on the application of blockchain technology for enhancing transparency and traceability in the cold chain. It facilitates the automatic execution or recording of relevant events under certain conditions through smart, digitally coded contracts. This technology helps in establishing trust, quality assurance, and identifying faults or inefficiencies in real-time (Cargo Insights’ 2023; IBM 2023).

The research on technological advancements in cold chain technology is notably sparse, largely due to the industry's secretive nature. Companies guard their innovations, limiting public and academic access to information about new technologies. This reticence results in a literature landscape dominated by general overviews rather than detailed analyses of recent advancements. The gap highlights the need for increased industry-academia collaboration and policies promoting transparency in the cold chain sector.

2.6. Micro-Controllers & Embedded Systems

Embedded systems and micro-controllers, crucial in advancing automation and smart technology, constitute programmable systems integral to various applications. Embedded systems are typically developed to have 1 specialized functionality in contrast to a general computer which can complete multiple tasks (Creative and Bluefruit 2022). Embedded systems usually incorporate either microprocessors or microcontrollers, which are integrated circuits that provide computational power to the system (IoT Agenda 2023).

Today's predominant use of microcontrollers is within various embedded systems, including vehicles, phones, household appliances, and computer system peripherals. They play a central role in enhancing the precision and efficiency of production processes across multiple industries. Furthermore, their capability to integrate with different sensor types underscores their versatility in meeting the growing needs for accuracy in industrial, medical, and technological domains. This adaptability not only optimizes operational outcomes but also significantly contributes to the ongoing development of sophisticated smart technology applications.

2.6.1. Technological Advancements & Current Trends

Over the years, the field of embedded systems has experienced remarkable technological advancements, significantly enhancing their capabilities and applications. In the early stages, embedded systems were rudimentary, often limited to simple control tasks due to their modest processing power and minimal memory as outlined above. For example, the Apollo Guidance Computer, which is widely considered as the first embedded system, helped guide Appollo astronauts on their journey to the moon (Hack the Moon 2023).

2.6.1.1. Wireless Connectivity/Technology

Connectivity has been another area of significant progress. Although some embedded systems operate in isolation (Car Rental Gateway 2020), the integration of networking capabilities, such as Wi-Fi, Bluetooth, Radio Frequency (RF) and cellular technologies, transformed them into interconnected, smarter systems (Promwad 2023). This connectivity laid the foundation for the Internet of Things, where embedded systems now play a critical role in enabling smart homes, industrial automation, and wearable technology (Evans 2023).

2.6.1.1. Power Consumption

As technological advancements continue in the realm of embedded systems, a key focus has been on enhancing power efficiency. The fundamental principle here is the lower the power consumption of the electronic components, the longer the device can operate before there's a need to recharge or replace its power source, such as a battery. This efficiency is particularly critical in applications where frequent recharging or battery replacement is impractical or impossible (Almusallam 2015).

Modern embedded systems have benefited from various innovations aimed at reducing power consumption. These include the development of low-power microcontrollers, energy-efficient sensors, and power management techniques like sleep modes and dynamic voltage scaling. Such advancements are not only extending the operational lifespan of these systems but also broadening their applicability in areas like remote sensing, wearable technology, and IoT devices, where power efficiency is crucial (Walls 2015).

2.7. Comparative Analysis of Micro-Controllers: ESP32, Arduino Uno R3, and Raspberry Pi Pico 2040

Below is a comparative analysis carried out on modern micro controllers. All micro controllers have different specifications and applications within the real word. The researcher intends to provide insight into 3 different types of micro-controllers. This comparative review is informed by the latest specifications and data sheets released by their respective manufacturers: Raspberry Pi Ltd (2023), Espressif Systems (2023), and the developers of the UNO R3 (2023) unless otherwise referenced.

2.7.1. Processing Capabilities

ESP32: Features a dual-core Tensilica LX6 microprocessor with a clock speed of up to 240 MHz, making it highly capable for multitasking and handling complex computations.

Arduino Uno: Powered by an ATmega328P microcontroller with a 16 MHz clock speed, it is suitable for simpler, less demanding tasks.

Raspberry Pi Pico: Utilizes a dual-core Arm Cortex-M0+ processor, clocked up to 133 MHz, offering a balance between the ESP32 and Arduino Uno in terms of CPU power.

2.7.2. Memory and Storage

ESP32: Equipped with 520 KB SRAM and up to 4 MB of external flash memory, it excels in applications requiring substantial memory management.

Arduino Uno: Offers 2 KB SRAM, 32 KB flash memory, and 1 KB EEPROM, sufficient for basic programs and data storage.

Raspberry Pi Pico: Comes with 264 KB of SRAM and 2 MB of onboard flash memory, providing a middle ground in memory capacity.

2.7.2. Connectivity

ESP32: Stands out with built-in Wi-Fi and Bluetooth capabilities, ideal for IoT applications requiring wireless communication. Has variants that feature Lo-Ra and an LCD (Liquid Crystal Display) screen.

Arduino Uno: Lacks native connectivity features but can be expanded with shields for networking capabilities.

Raspberry Pi Pico: Like Arduino Uno, it requires additional modules for connectivity.

2.7.3. Power Consumption

ESP32: Its advanced features like WI-FI and Bluetooth lead to higher power consumption, making it less ideal for battery-powered applications (Last Minute Engineers 2018).

Arduino Uno: Known for its power efficiency with an average power consumption of 734mW (Lextrait 2016).

Raspberry Pi Pico: Also, power-efficient, with an average of 86.5 mA (‘Raspberry Pi Pico Datasheet).

2.7.4. Ease of Use

ESP32: Offers flexibility with programming options like the Arduino integrated development environment (IDE) which supports C/C++ and ESP-IDF, among others.

Arduino Uno: Supports both C/C++ , beginner-friendly with extensive community support and a plethora of educational resources.

Raspberry Pi Pico: Supports both C/C++ and Micro Python, appealing to a broad range of users, with growing community support.

2.7. Security in Micro-Controllers & Embedded Systems

In the realm of microcontrollers and embedded systems, security has emerged as an area for major concern, especially given their extensive application in critical domains ranging from automotive systems to healthcare devices. The ubiquity of these systems, coupled with their increasing connectivity, elevates the importance of robust security measures (Wilcocks 2023).

Embedded systems often operate with constraints such as limited processing power, memory, and energy resources, posing unique challenges for implementing comprehensive security protocols. Moreover, the diverse nature of these systems, encompassing various hardware configurations and software platforms, complicates the standardization of security measures. Additionally, the long operational lifespans of many embedded devices can leave them vulnerable to evolving cybersecurity threats (Savjani 2018).

2.7.1. Security Threats and Vulnerabilities

The primary security threats to embedded systems and microcontrollers include firmware tampering, unauthorized data access or interception, and replication attacks. Firmware tampering can alter system functionality, while data breaches threaten confidentiality and integrity. Furthermore, the physical accessibility of many embedded systems introduces additional risks of direct tampering and exploitation (Jasani 2021; Savjani 2018).

2.7.2. Strategies for Enhancing Security

To mitigate these threats, several strategies can be employed (Jasani 2021; Avila 2023; Media 2023):

Secure Boot Mechanisms: Ensuring the integrity of the system at boot time is crucial. Secure boot processes validate the authenticity and integrity of the software running on the device.

Encryption: Encrypting data stored and transmitted by embedded systems protects against unauthorized access and interception of potentially sensitive & confidential data.

Regular Updates: Keeping firmware and software updated is essential to patch known vulnerabilities.

Access Control and Authentication: Implementing robust authentication mechanisms prevents unauthorized system access.

Intrusion Detection Systems: Monitoring for abnormal activities helps in early detection and response to potential security breaches.

2.8. Sensors in Cold Chain Pharmaceutical Monitoring Systems

In the context of cold chain management for pharmaceuticals, the application of advanced sensor technologies is indispensable for ensuring the integrity and efficacy of temperature-sensitive products. These sensors, integral to the monitoring and control systems, play a crucial role in providing real-time data and maintaining the required environmental conditions during storage and transportation. This part of the literature review delves into the various types of sensors employed in the pharmaceutical cold chain, highlighting their specific functions and importance.

2.8.1. Temperature Sensors

In the domain of pharmaceutical cold chain management, the utilization of temperature sensors represents a critical aspect for ensuring the integrity and efficacy of temperature-sensitive pharmaceutical products. These sensors come in different both wireless and wired, some of these include but not limited to (Sensitech 2023):

Ambient Temperature: These sensors measure the environmental temperature surrounding the pharmaceutical products. They are crucial in ensuring that storage and transport environments are within the required temperature ranges.

Infrared Sensors: Non-contact infrared sensors are used to remotely measure the temperature of a product or environment. They are useful in situations where direct contact with the product is not possible or might lead to contamination.

Temperature Probes: Temperature probes are instruments designed to gauge temperature through contact-based sensing techniques. These techniques determine temperature by employing probes that detect variations in a temperature-responsive property, such as voltage difference or resistance (GlobalSpec 2023).

Cryogenic sensor: Cryogenic sensors also known as Cryos are specialized temperature sensors designed to operate at extremely low temperatures, often in the range of cryogenic temperatures (below -150°C or -238°F) (Processing 2020). These sensors are crucial in environments where standard temperature sensors cannot function accurately due to the extreme cold. In the pharmaceutical industry, cryogenic sensors are particularly important for monitoring the storage conditions of certain drugs and biological materials that require ultra-low temperature environments. This includes some vaccines like the “Pfizer-BioNTech COVID-19” vaccine that first came to the market during the covid-19 pandemic (LabRepCo n.d.) and other biological samples that must be kept at cryogenic temperatures to maintain their stability and efficacy. Cryogenic sensors ensure precise and reliable temperature monitoring in these critical applications (lakeshore.com 2023).

2.8.2. Oxygen Sensors

Oxygen sensors are crucial in ensuring both personnel safety and the integrity of valuable inventory in confined spaces such as freezers or cold storage areas. They serve as an early warning system against oxygen depletion, which can occur due to leaks or equipment malfunction. Low oxygen levels pose a serious asphyxiation risk to personnel, making these sensors vital for alerting individuals to dangerous conditions, allowing for prompt evacuation or precautionary measures. Additionally, in environments storing perishable goods, such as food products or pharmaceuticals, maintaining specific oxygen levels is essential for preserving product quality and shelf life. Oxygen sensors help detect any potential depletion issues early, enabling timely corrective actions to protect the stored inventory, thereby safeguarding both human safety and the quality of critical goods (CO2 Meter 2023).

2.8.3. Carbon Dioxide Sensors

In pharmaceutical storage and transportation, where temperature control is typically the primary concern, the role of Carbon Dioxide (CO2) sensors is increasingly recognized for certain critical applications. These sensors are particularly vital in scenarios where maintaining controlled atmospheric conditions is as essential as regulating temperature. A key example is the transportation of pharmaceuticals using dry ice as a cooling agent. CO2 sensors are crucial here for monitoring CO2 levels to ensure both the effectiveness of the cooling process and safety, as excessive CO2 can pose hazards to transport personnel. CO2 sensors are indispensable in the storage of specialized pharmaceutical products that require not only specific temperature conditions but also controlled atmospheric environments. Precise CO2 level maintenance is essential for the stability and efficacy of these products, making continuous monitoring and regulation of CO2 concentrations a critical aspect of their storage (Vaisala 2021).

2.8.4. Humidity Sensors

Humidity sensors play a critical role in pharmaceutical cold chain management, ensuring the maintenance of optimal moisture levels during the storage and transport of sensitive pharmaceutical products. Excessive or insufficient humidity can adversely affect the stability, efficacy, and shelf life of various medications. These sensors are particularly important for hygroscopic drugs, which are sensitive to moisture. By continuously monitoring and regulating humidity, these sensors help in adhering to stringent storage guidelines and preventing product degradation. Accurate humidity control, facilitated by these sensors, is essential for maintaining product quality and ensuring compliance with regulatory standards in the pharmaceutical industry (Neutroplast 2023; medsage.gov.nz 2023).

2.8.5. Door Sensors

Door ajar sensors have a crucial role in ensuring the integrity of temperature-controlled environments. These sensors, typically employing magnetic or mechanical switches, function by completing a circuit when a door is securely closed. Upon opening, the circuit breaks, initiating an alert. This mechanism is essential in signalling the opening of doors to refrigerated storage areas or transport vehicles, thereby preventing accidental exposure of sensitive pharmaceutical products to inappropriate temperature and humidity levels. Configurable to various alert systems, these sensors can activate wireless IoT alarms, RF Alarms, or simple visual indicators like light-emitting diodes (LEDs), ensuring immediate awareness and prompt response to maintain optimal storage conditions (Habas 2023; '#01 What's a Magnetic Sensor?' 2023; 'Door Open Sensors: Applications for Fleets' 2023).

2.8.6. Regulatory Compliance and Standards for Sensors in Cold Chain Monitoring

Devices that rely on sensors, especially those used in healthcare, need to adhere to certain regulatory standards to guarantee they are safe for users in terms of electrical, chemical, biological, and physical aspects. The extent of regulation varies based on the device's associated risk level. For instance, implantable devices like pacemakers are subject to stricter regulations compared to non-invasive tools such as thermometers. It is crucial for developers and users of health, wellness, or environmental devices to understand and comply with the specific regulations relevant to their devices (McGrath and Scanaill 2013).

Regulations often stipulate the types of sensors that can be used, their calibration requirements, and the frequency of recording sensor data. The standardization of sensor technology in accordance with these regulations ensures consistency in monitoring across the industry. It also guides pharmaceutical companies in selecting appropriate sensor systems that are compliant with regulatory standards (Mikeej 2021; Gemini Data Loggers 2023).

2.8.7. Evolving Role of IoT Sensors

According to Newton (2021) utilizing IoT for the cold chain market is growing fast and is estimated to be worth more than $4.79 billion in 2021. These sensors, integral to the Internet of Things ecosystem, offer sophisticated capabilities beyond traditional monitoring methods.

IoT sensors in cold chain monitoring are designed to provide real-time tracking of critical parameters like temperature and location at once (Inc 2022). Their ability to connect to the internet or a network with cellular without the need for a controller allows for continuous data transmission, enabling stakeholders to monitor the status of pharmaceutical products remotely and in real-time (Velos 2023). This connectivity is crucial for maintaining the integrity of temperature-sensitive products throughout the supply chain.

2.9. Conclusion

In conclusion, this chapter thoroughly explores the intersection of machine learning, microcontrollers, embedded systems, and sensor technologies in pharmaceutical cold chain monitoring. However, a significant research gap is evident, particularly in the practical integration of these technologies for optimized cold chain management. While advancements in machine learning algorithms like ARIMA, RNNs, and LSTM show promise for predictive modelling and data analysis, their real-world application in the pharmaceutical sector remains under-explored.

Similarly, while the chapter details the evolution and potential of microcontrollers and embedded systems, there is a lack of comprehensive research on their most effective implementation in pharmaceutical logistics. Security in these systems is another area where more in-depth studies are required to address the unique challenges of the pharmaceutical sector.

The role of sensor technologies in monitoring temperature, humidity, and other critical parameters is well-established, yet there is a dearth of comparative studies, particularly between RF and IoT sensors, to determine the most effective solutions for both storage and transport in the pharmaceutical cold chain.

Finally, while the chapter touches on regulatory compliance and the evolving role of IoT sensors, there is a clear need for more focused research on integrating these technologies in a manner that meets regulatory standards and optimizes the entire cold chain process. This research gap underscores the need for further studies that not only delve into the theoretical aspects of these technologies but also their practical, real-world applications in the pharmaceutical cold chain, aiming to enhance efficiency, reliability, and compliance in this critical sector.

3. Design and Methodology

3.1. Introduction

This chapter is dedicated to outlining the comprehensive design framework of the system under development in this project. It delves into both the software and hardware prerequisites, shedding light on the details of each. Additionally, the chapter will delve into the functional and non-functional requirements that are fundamental to the system's operation. Emphasis will be placed on illustrating key design elements through various diagrams & schemas. Furthermore, this chapter will illustrate the development process of critical components, including the construction of machine learning models and the embedded system. The insights and theoretical underpinnings acquired from the literature review, particularly those concentrating on machine learning applications & micro-controller driven sensor systems and methodologies, will shape the design strategy adopted in this chapter.

3.2. System Requirements

3.2.1. Software Requirements

3.2.1.1 Python

For the project, Python will be exclusively utilized for the machine learning aspect, particularly in training, developing, and testing the model for sensor data analysis. This approach capitalizes on Python's robust capabilities in data processing and machine learning, with its rich libraries like Pandas, NumPy, and machine learning frameworks such as TensorFlow and scikit-learn. These tools are essential for handling complex data manipulations and constructing efficient models for anomaly detection and time series analysis.

3.2.1.2 C++

C++ is particularly suitable for programming the ESP32 in this project due to its efficiency and performance in embedded systems. The language's low-level control allows for optimized handling of the ESP32's limited processing and memory resources, essential for real-time sensor data collection. Additionally, C++ provides robust library support for interfacing with hardware, making it ideal for managing the various sensors in the system. This efficiency and control make C++ an excellent choice for the embedded component of the project, ensuring reliable and responsive performance.

3.2.1.3 Google Colab

Google Colab is a cloud-based platform that allows users to write and execute Python code through their browser. It is widely recognized in the machine learning community for its ease of use, zero-configuration setup, and free access to powerful computing resources, including GPUs and TPUs. Colab is built on top of Jupyter Notebook, an open-source web application that is well-regarded for its ability to create and share documents containing live code, equations, visualizations, and narrative text.

3.2.1.4 MySQL Workbench 8.0 IDE

MySQL Workbench will be employed for data preprocessing in this project, leveraging its sophisticated yet user-friendly interface for managing and manipulating the sensor data. As a comprehensive IDE for MySQL, it excels in facilitating various data operations essential for preparing data for machine learning. With MySQL Workbench, the researcher can efficiently perform tasks such as querying, filtering, and transforming sensor data, ensuring that the data is clean, well-structured, and optimized for analysis.

3.2.1.5 Arduino IDE

For this project, the Arduino IDE will be utilized specifically for the development of the ESP32 microcontroller. Known for its user-friendly interface and wide compatibility with various microcontrollers, the Arduino IDE is an excellent tool for programming the ESP32. It provides a straightforward environment for writing, debugging, and uploading code to the ESP32, facilitating efficient development of the firmware needed for sensor data management and processing. This choice capitalizes on the Arduino IDE's ease of use and robust community support, ensuring a smooth development process for the ESP32's role in the project.

3.2.1.6 Flask

For this project, Flask is selected as the web framework of choice due to its lightweight and adaptable nature, perfectly suited for building efficient RESTful Application Programming Interfaces (API) for data management and representation. Its seamless compatibility with Python enhances its integration with a vast array of data processing libraries, essential for the pre-processing needs in machine learning workflows. Flask's straightforward yet powerful functionality allows for quick development and deployment. Moreover, its extensive community support and comprehensive documentation ensure a smooth development process, making Flask an excellent choice for efficient and effective backend service implementation in this sensor-data-driven project.

3.2.2. Hardware Requirements

3.2.2.1 Micro-Controller Components

Depicted in table X is a list of components and their respective costs that are required to assemble the micro-controller.

Table 1 - Micro-Controller Components and Their Cost

Component	**Cost**
ESP32 - DevKitC	€10
Micro SD Card Memory Shield Module	€2
Resistors	€1
NRF24L01 - 2.4G Wireless Transceiver Module	€5
LCD Display Screen	€8
Battery Holder - 8xAA - 12V	€8
LED - Basic Yellow	€0.50
LED - Basic Green	€0.50
LED - Basic Red	€0.50
Heatsink TO-220	€1
Total:	€36.50

The selection of components for assembling the micro-controller has been strategically made with an aim for cost-effectiveness without compromising on functionality and performance. The total cost amounts to €36.50, which is remarkably economical for a comprehensive set of components required for a versatile and efficient micro-controller system.

3.2.2.2 Sensors

To augment the Micro-Controller's functionality, the system employs a suite of sensors, each providing essential data for representation on a user-focused web interface Notably, the first two sensors are being provided by Kelsius for research purposes:

RF Temperature Sensor: This sensor offers real-time temperature data, leveraging wireless communication for seamless integration with the Micro-Controller, essential for applications requiring precise temperature monitoring.

RF Humidity Sensor: It measures ambient humidity with high accuracy, vital for applications sensitive to environmental conditions, and wirelessly transmits this data for analysis and display.

RuuviTag IoT Sensor: A versatile IoT device, the RuuviTag captures not only temperature and humidity but also air pressure, providing a comprehensive environmental overview, crucial for nuanced data-driven insights and control.

3.2.2.3 Laptop

The selected laptop, equipped with specifications tailored to meet the software requirements, will be utilized for this project.

Processor: 11th Gen Intel (R) Core (TM) i7-1185G7 @ 3.00GHz   1.80 GHz

Installed RAM: 16.0 GB DDR4

System type: 64-bit operating system, x64-based processor

SSD: 240GB

3.2.2.4 Desktop

The chosen desktop, which possesses the following specifications, has been identified as suitable for hosting the web component, as it aligns with the hosting requirements necessary for the project.

Processor: 9th Gen Intel (R) Core (TM) i9-9900K CPU @ 3.60GHz   3.60 GHz

Installed RAM: 16.0 GB DDR4

System type: 64-bit operating system, x64-based processor

Graphics Card: Nvidia GTX 4060

SSD: 1TB

3.3. Application Requirements

3.3.1. Functional Requirements

A Functional Requirement is a specification that describes what a system, process, or product must do. It defines specific functionalities or services that must be provided to meet the user's needs and achieve the desired outcome.

FR1: Data Collection

The system must be able to collect temperature data from various wireless sensors connected to the microcontroller.

FR2: Secure Data Transmission

The system should transmit data securely from the microcontroller to a centralized server or cloud-based system for further analysis.

FR3: Machine Learning Integration

The system must integrate machine learning algorithms for anomaly detection and time-series forecasting of temperature data.

FR4: Real-Time Monitoring

The system must continuously monitor temperature data in real-time, ensuring immediate detection of any deviations from the set thresholds. It should automatically log all temperature readings with timestamps.

FR5: Web-Based Interface

The application must have a web-based interface that allows users to view real-time data, run reports, and receive alerts on temperature deviations.

FR6: Sensor Data Evaluation

The system should be capable of evaluating and comparing data from different types of sensors, specifically RF and Wi-Fi-based sensors, to determine their suitability.

FR7: Testing Framework for Machine Learning Models

The system shall include a framework for testing and validating machine learning models, ensuring their accuracy and reliability against test datasets.

FR8: Secure Communication Protocols

The system must have secure communication protocols like SSL/TLS for data in transit and WPA3 for wireless communication to ensure data security and privacy.

FR9: Automated Alert System

An automated alert system that notifies users of any detected anomalies or significant temperature deviations.

3.3.2. Non-Functional Requirements

Non-Functional Requirements (NFRs) are specifications that describe the system's quality attributes, performance metrics, and constraints. Unlike Functional Requirements that define what a system should do, NFRs outline how a system should behave and under what conditions, ensuring the system's reliability, efficiency, and usability.

NFR1: Accuracy and Consistency

The system must deliver temperature readings with a minimum accuracy of 99% and ensure system downtime does not exceed 30 minutes per week, guaranteeing reliable and consistent data collection critical for effective analysis.

NFR2: Scalability

It should be scalable to accommodate an increasing number of sensors and larger datasets.

NFR3: Real-Time Data Processing

The system should process and analyse data in real-time or near-real-time to enable prompt decision-making.

NFR4: User-Friendly Web Interface

The web interface should be user-friendly, allowing users with varying levels of technical expertise to easily navigate and use the system.

NFR5: Security Measures

Security measures must be in place to protect sensitive data from unauthorized access and cyber threats.

NFR6: Compliance with Data Privacy Laws

Compliance with data privacy laws and regulations, ensuring that user data is handled confidentially.

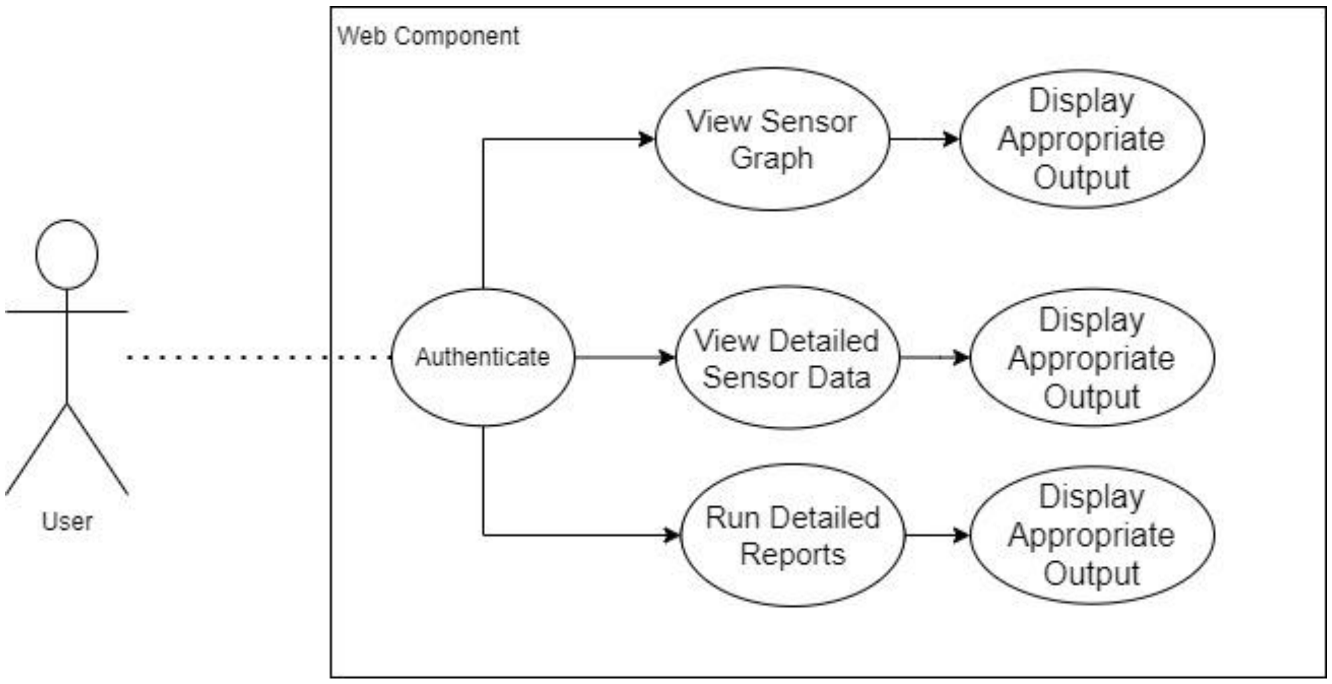
NFR7: Energy Efficiency

The microcontroller and sensors should be optimized for low power consumption, considering the energy constraints in healthcare settings.

3.4. Use Case Diagram

Figure 5 presents a Use Case Diagram, illustrating the interaction between the user and the web component of the system. This diagram serves as a visual guide, detailing the various ways in which users engage with the web interface.

Figure 5 – Use Case Diagram for Web Component


Web Component
User
Authenticate
View Sensor Graph
View Detailed Sensor Data
Run Detailed Reports
Display Appropriate Output
Display Appropriate Output
Display Appropriate Output

3.5. Use Case Tables

Below are detailed use case tables for the functionalities of the web component. Each table outlines a specific use case, providing a comprehensive view of the objective, preconditions, main flow of events, alternative flows, and postconditions. These use cases are designed to give a clear understanding of the system's functionalities and user interactions.

Table 2 – Use Case Table for Login

Use Case	**Login**
Objective	To authenticate the user, providing access to the system's monitoring and reporting features, thereby ensuring data security.
Precondition	User must be registered with valid credentials.
Main Flow	1. User navigates to the login page. 2. User enters their credentials. 3. System validates the credentials. 4. User is granted access to the system.
Alt Flow	If credentials are invalid, the system displays an error message and prompts the user to try again.
Post Condition	User gains access to the system and can use its features.

Table 3 – Use Case Table for Sensor Graph

Use Case	**View Sensor Graph**
Objective	To allow the user to view and interact with sensor data, including selecting different time ranges and sensors.
Precondition	User must be logged into the system.
Main Flow	1. User selects the Sensor Graph option. 2. User chooses a sensor and time range. 3. System displays the graph based on the selected parameters.

Alt Flow	User may switch between different sensors or adjust time ranges as needed.
Post Condition	User can view and interact with the sensor data graph.

Table 4 – Use Case Table for Running Reports

Use Case	**Run Detailed Reports**
Objective	To enable the user to generate detailed reports for proactive analysis based on sensor data.
Precondition	User must be logged in and viewing sensor data.
Main Flow	1. User selects the option to generate a report. 2. User specifies criteria for the report. 3. System processes and generates the report.
Alt Flow	User may modify criteria or cancel the report generation process.
Post Condition	A detailed report is generated and available for the user to view.

Table 5 – Use Case Table for Viewing Details Sensor Data

Use Case	**View Detailed Sensor Data**
Objective	To provide a visual representation of temperature data over time, including actual and predicted values, with alerts for threshold exceedances.
Precondition	User is logged in and has selected a specific sensor graph.
Main Flow	1. User accesses a detailed view of a selected sensor graph. 2. System displays temperature data over time, including predictions. 3. System marks points where thresholds are exceeded.
Alt Flow	User may zoom in/out or adjust the time scale for more detailed or broader views.

Post Condition	User gains insights into temperature trends, predictions, and receives alerts for critical changes.

3.4. System Architecture

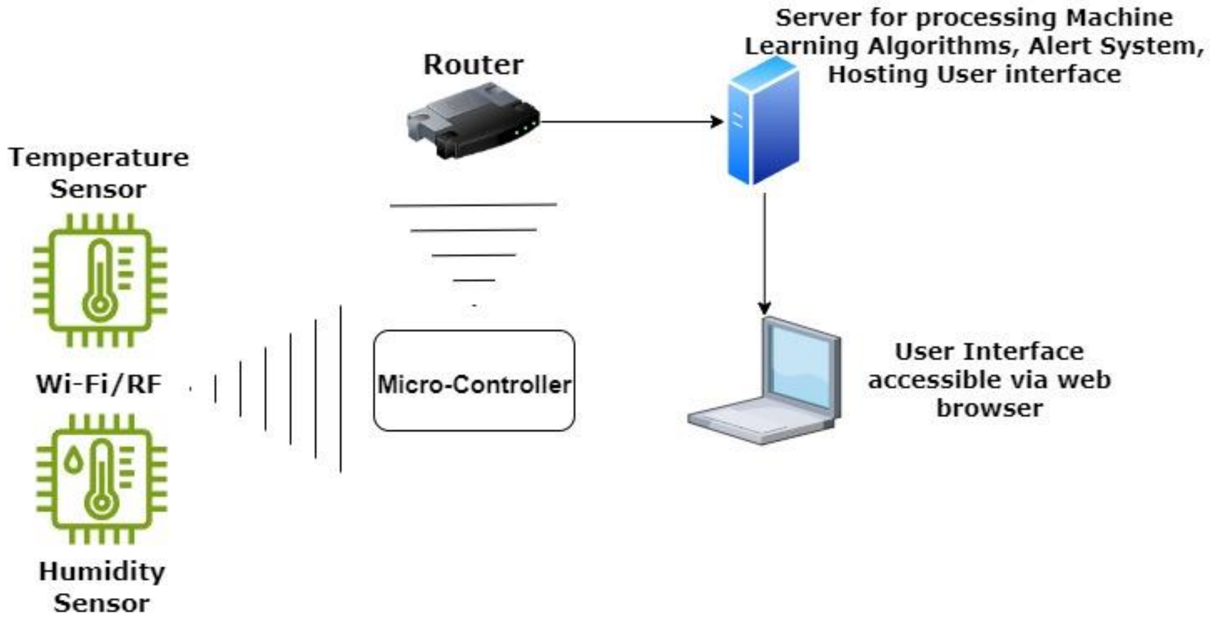

Server for processing Machine Learning Algorithms, Alert System, Hosting User interface
Router
Temperature Sensor
Wi-Fi/RF
Micro-Controller
User Interface accessible via web browser
Humidity Sensor

Figure 6 – High Level Diagram of The System Architecture

The system architecture depicted in Figure 6 represents the proposed solution that integrates various sensors for data collection, transmission, processing, and user interaction. Below is a detailed description of the System Architecture based on the above Figure.

Sensors: At the base level, there are two types of sensors—temperature and humidity sensors. These are likely equipped with Wi-Fi/RF capabilities for wireless data transmission. The sensors' role is to collect environmental data accurately and reliably.

Micro-Controller: The sensors interface with a microcontroller, which acts as a data aggregator and controller. The microcontroller is responsible for initial data processing. It also manages the communication between the sensors and the next level in the architecture.

Router: The router serves as a gateway for data transmission from the micro-controller to the internet and finally the server. This component ensures that data packets are sent securely and efficiently to the server over the network.

Server: At the higher level is the server. Its functions are multi-fold. It runs machine learning algorithms that analyse the temperature and humidity data for anomaly detection and time-series forecasting. It is responsible for the alert system, notifying users of any detected anomalies or significant deviations. It hosts the user interface, making the data and tools accessible like predictive analysis accessible to the users.

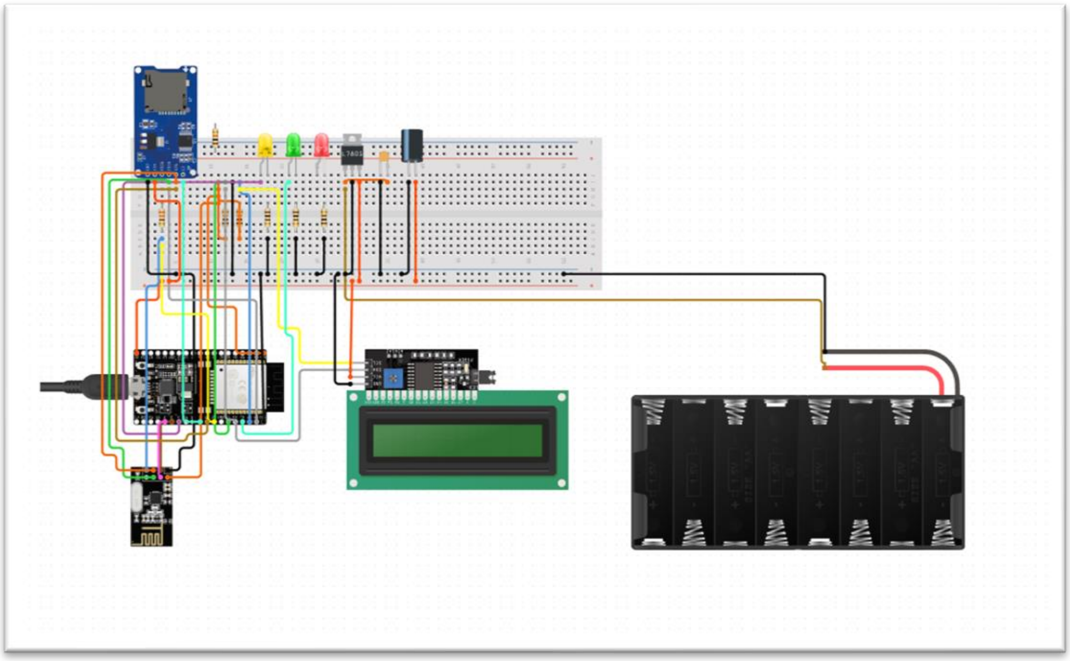
User Interface: The user interface is accessible via a web browser, indicating that it's a web-based application. Users can view real-time data, predicted data, run reports, and receive alerts through this interface.

3.5. Circuit Diagram

The circuit diagram in Figure 7 on the next page presents a view of the proposed microcontroller-based sensor system designed for temperature and humidity data acquisition and wireless communication. At the heart of the system is the ESP32-DevKitC, the researcher chose this controller as it is a versatile and powerful microcontroller with built-in Wi-Fi capabilities and costs a reasonable €10. The microcontroller interfaces with a variety of components, each serving a specific function within the system.

Some of these modules include a LCD screen, which provides a simple interface for real-time data output. The integration of a Secure Digital (SD) Card Memory Shield Module allows for expansive data logging and storage capabilities in case there is a failure on the network.

Figure 7 – Proposed Circuit Diagram of the Micro-Controller Responsible for Communication of The Sensors and the Network



Wireless RF communication is facilitated by the NRF24L01 - 2.4G Wireless Transceiver Module, this module was selected as it ensures efficient data transmission over the 2.4GHz frequency, allowing the system to communicate with the RF Sensors selected by the researcher.

The circuit is powered by an 8xAA Battery Holder, supplying 12V, ensuring that the system can operate independently of a fixed power source in case of a power failure.

Visual indicators are provided by three LEDs (yellow, green, and red) to signify different system states, such as power on, Sleep state for power saving, operational status, or alerts for threshold breaches. Overall, these components are reasonably priced at €36.50 making it an extremely cost-effective system.

3.6. Story Boards

3.5.1. Login Interface

Figure 8 portrays the Log in Story Board, which is the gateway to the system, requiring user authentication to access the monitoring and reporting features, thus ensuring data security.

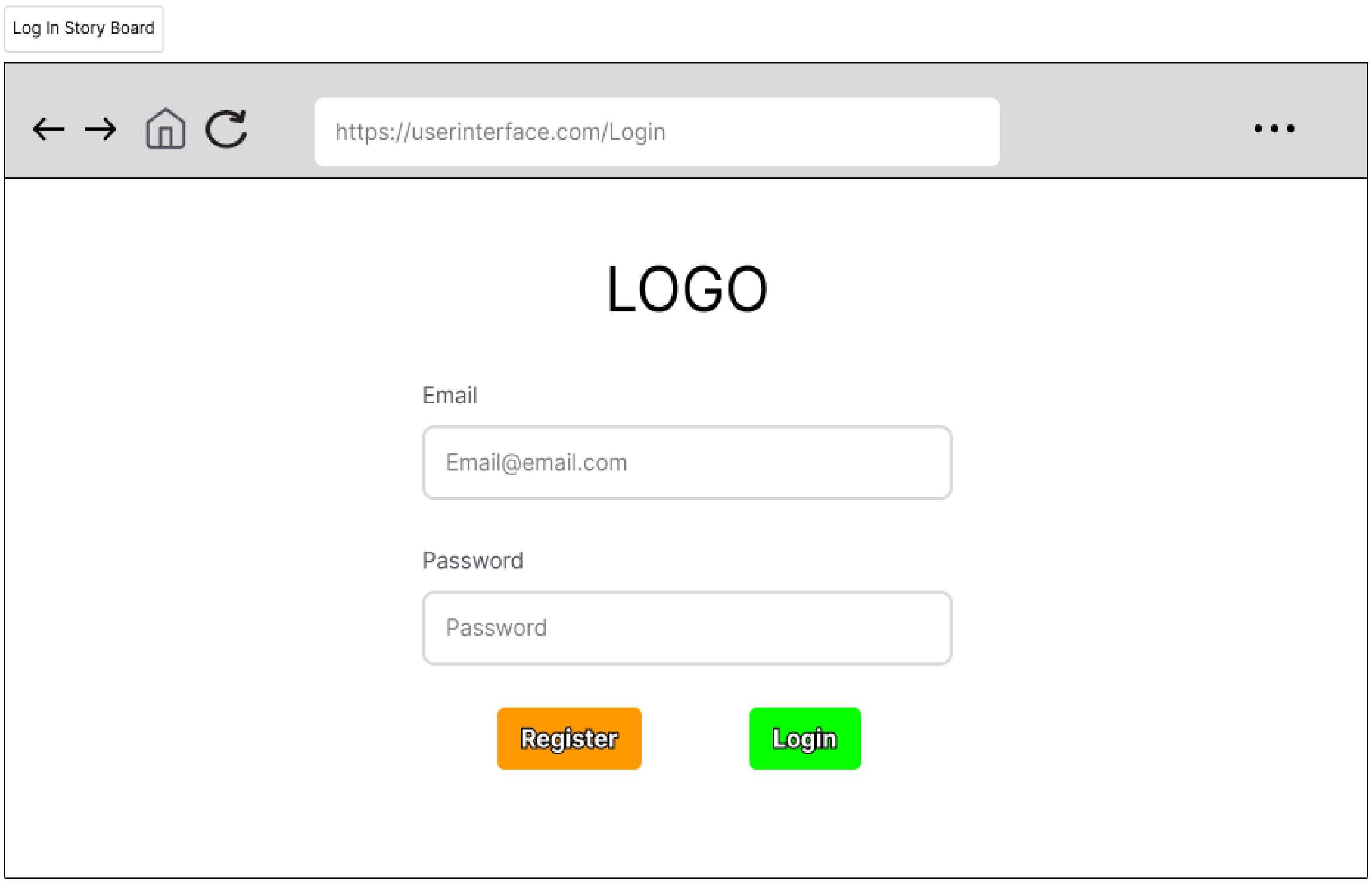
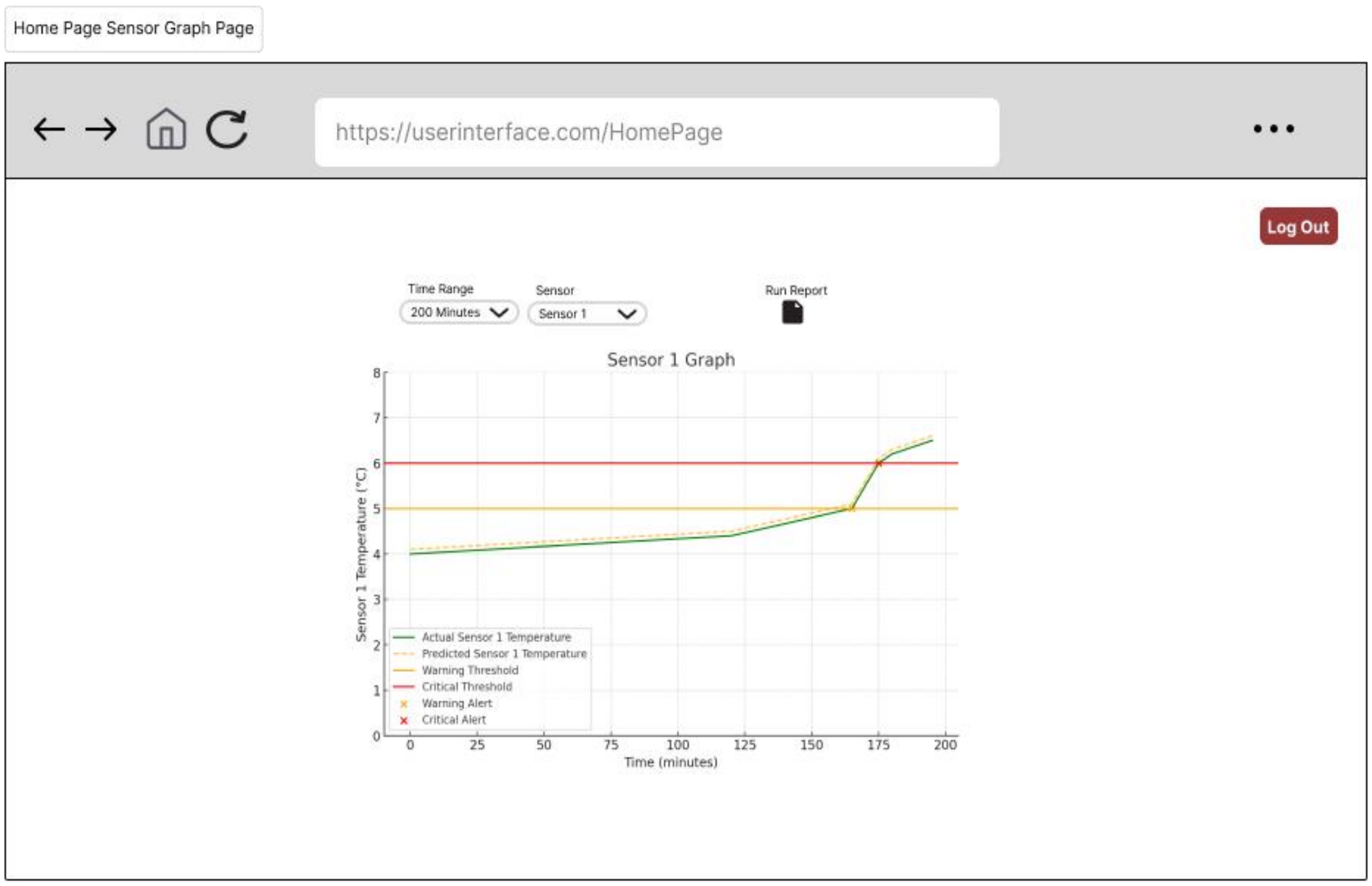


Figure 8 – Log In Story Board

3.6.2. Sensor Graph Interface

As shown in Figure 9 the webpage layout incorporates the Sensor 1 graph into a broader user interface, allowing users to select different time ranges and sensors for data display. It includes functionality to run detailed reports, these features allow for proactivity.



Home Page Sensor Graph Page
https://userinterface.com/HomePage
Log Out
Time Range
200 Minutes
Sensor
Sensor 1
Run Report
Sensor 1 Graph
Sensor 1 Temperature (°C)
Time (minutes)
Actual Sensor 1 Temperature
Predicted Sensor 1 Temperature
Warning Threshold
Critical Threshold
Warning Alert
Critical Alert

Figure 9 – Home Page Story Board

3.6.3. Sensor Graph in Detail

Figure 10 illustrates the Graph Storyboard, which provides a visual representation of temperature data over time, capturing both actual and predicted values. It also highlights alerts by marking a X when the hard coded thresholds (that are marked in green and orange in the below figure) are exceeded. This is crucial for the user to ensure efficacy of the contents within the environment.

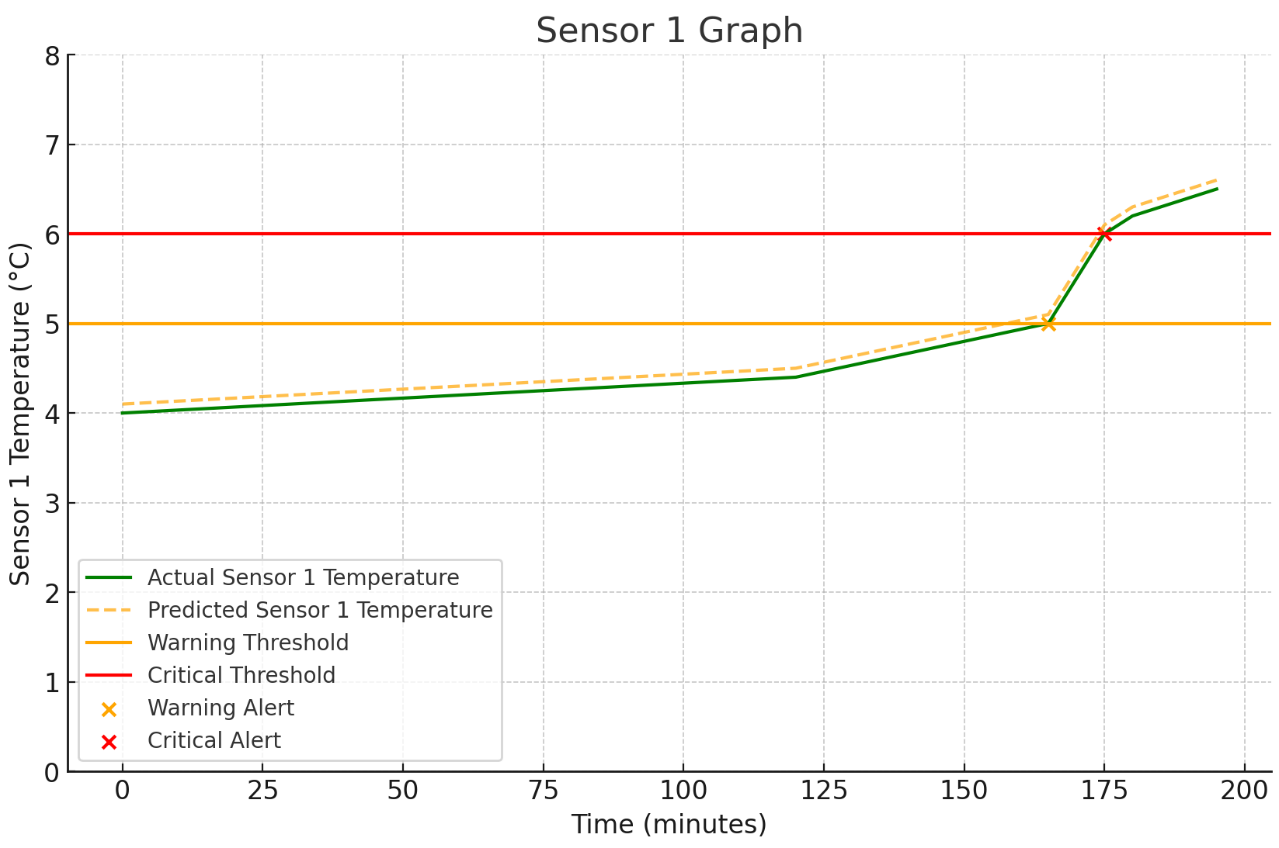


Figure 10 – Sensor Graph Story Board

3.6.4. Micro-Controller Interface

As shown in Figure 11, the Controller Interface storyboard presents the system's primary interface, displaying real-time temperature and humidity data from the sensors, which is essential for immediate oversight and monitoring.

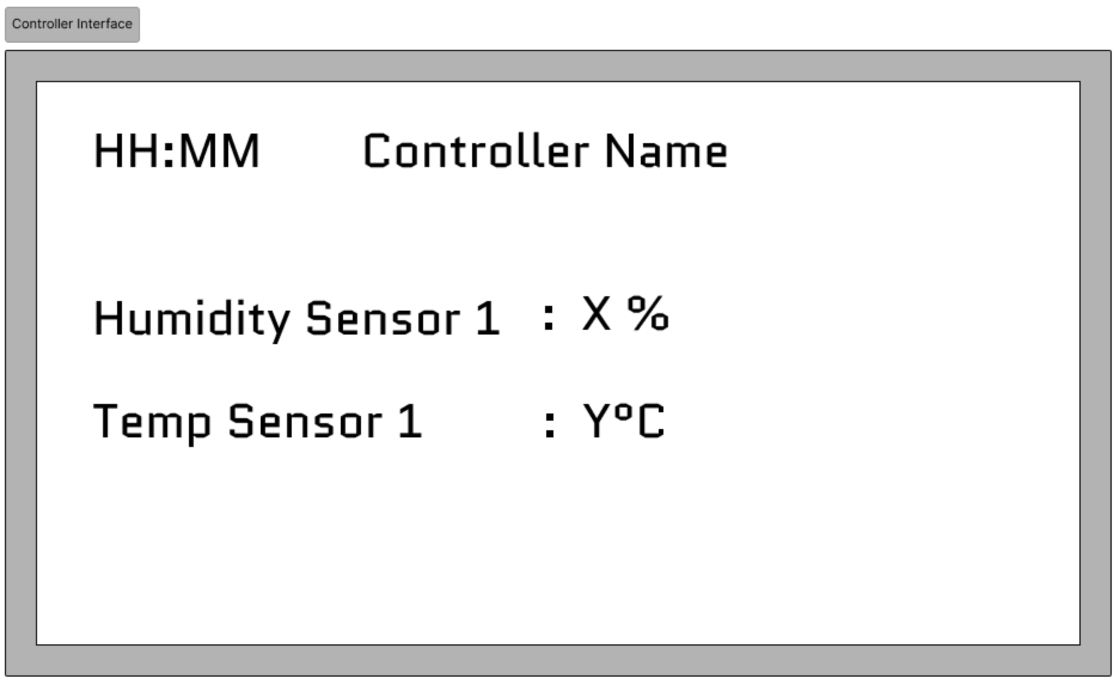


Figure 11 – Network Controller Interface Story Board

3.7. Data

3.7.1. Data Acquisition

Addressing the cold start challenge, the models require a substantial amount of data from the beginning. Kelsius, has been kind enough to provide access to their historical temperature data for this research. This data comes in the form of raw .SQL files, which requires preprocessing. The access to significant historical data is pivotal for training and calibrating the machine learning models in any sensor system of a similar nature, this ensures a solid foundation for accurate and effective real-time monitoring from the get-go.

3.7.2. Data Pre-Processing

To effectively utilize the temperature data provided by Kelsius for machine learning models, it is essential to first preprocess this data. Preprocessing will involve cleaning and normalizing raw data to ensure it is in a usable format for the models. This step is crucial for removing any inconsistencies or errors and for aligning the data with the specific requirements of the machine learning algorithms that are being employed. Through careful preprocessing, the data's reliability and accuracy in predictive modelling and analysis will be significantly enhanced. Figure 12 on the next page is a pipeline which demonstrates the steps that are required during the data pre-processing process.

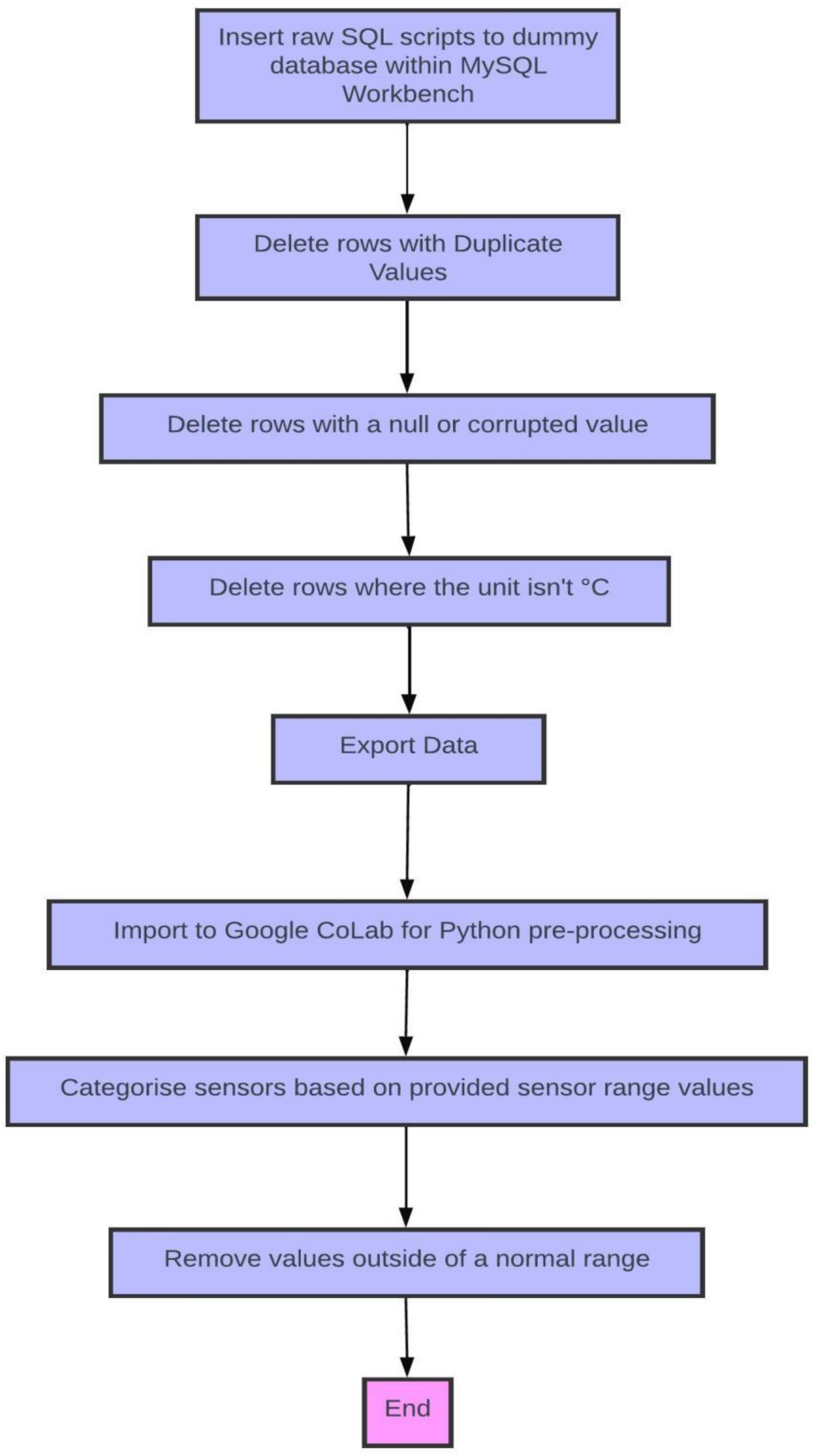

Insert raw SQL scripts to dummy database within MySQL Workbench
Delete rows with Duplicate Values
Delete rows with a null or corrupted value
Delete rows where the unit isn't °C
Export Data
Import to Google CoLab for Python pre-processing
Categorise sensors based on provided sensor range values
Remove values outside of a normal range
End

Figure 12 – Pipeline Which Demonstrates Steps That Are Required During the Data Pre-Processing Process

3.7.3. Model Selection

Selecting the appropriate machine learning models is a critical step in the development of the research. Building on the insights gained from the literature review, as analysed in chapter two, the researcher has decided to employ two distinct models for different aspects of temperature monitoring.

The ARIMA model will be utilized for predicting future temperature trends. ARIMA's strengths in time series forecasting make it well-suited for anticipating temperature changes, which is vital in maintaining the integrity of cold chain pharmaceuticals.

Secondly, for classification and anomaly detection, the LSTM model will be employed. LSTM, a type of recurrent neural network, excels in recognizing patterns over time, making it ideal for identifying irregularities in temperature data that could indicate potential issues.

In practice, the ARIMA model informs about expected temperature trends, setting a baseline for normal conditions. The LSTM model then continuously analyses incoming data against this baseline to flag any anomalies. This collaborative approach allows for both effective prediction and vigilant monitoring of temperature conditions.

3.7.4. Dividing Data & Training the Model

After selecting the machine learning models, the next step involves dividing the pre-processed temperature data into training and testing sets. The training set will be used to teach the models to recognize and predict temperature trends (ARIMA) and identify anomalies (LSTM). A significant portion of the data is allocated for training. The remaining forms the testing set, which is crucial for evaluating the models' performance on data they haven't seen before. This testing ensures that the models are accurate and reliable in real-world scenarios. During this phase, various parameters of the models are adjusted to optimize their performance. After training, the models are then validated and tested using the testing set to assess their accuracy and effectiveness in temperature prediction and anomaly detection.

3.7.5. Data Representation

After data preprocessing and model training in Google Colab, the outcomes are seamlessly integrated into a Flask-based web application. This setup harnesses Google Colab for complex computations and efficiently transitions the data for interactive visualization using Python's powerful libraries. Users can access and interpret real-time and predicted temperature trends, including any anomalies, through this application. The integration of Python libraries in Flask ensures a robust and user-friendly platform, making the project's results both technically sound and easily accessible for practical use.

3.8. Web Component

This section of the design chapter will concentrate on crafting a web component using Flask, a Python-based web framework known for its lightness and versatility. Flask's compatibility with Google Collab makes it an ideal choice for this project. The core aim here is to develop a straightforward authentication website, ensuring secure user access to their data. This site will boast a user-focused design, enabling authenticated individuals to view graphs corresponding to each sensor linked to their controller.

The selection of Flask as the framework is due to its straightforward and effective nature in web application development. It provides essential functionalities for establishing a robust authentication system, safeguarding sensor data access to authorized users only. The website is envisioned to offer a clean, user-friendly interface, simplifying the process of navigating and visualizing data through graphical representations from each sensor.

These visualizations will be dynamically rendered, reflecting real-time data from the sensors. This approach not only elevates the user experience through meaningful visual data display but also assists in promptly spotting trends and irregularities in sensor measurements.

Moreover, the website will feature a mechanism for users to create and download a report of historical temperature data. This functionality enables users to gather and export past data in a well-organized, portable format, enhancing record-keeping and data sharing. The design of the report generation tool will emphasize ease of use, allowing users to select specific time periods and sensors for their reports.

Incorporating Flask into this web component is a strategic move, aligning with the overarching objective of establishing a functional, intuitive, and secure interface. This interface will facilitate user engagement with their sensor data and support the documentation and analysis of historical data patterns.

3.9. Sensor Configuration

This design will focus on integrating various digital RF sensors provided by Kelsius into the implementation of the system. These sensors capable of measuring temperature and humidity, are crucial for accurate and efficient temperature reading. They connect to the ESP32 microcontroller through an installed RF module, facilitating wireless data transmission. Additionally, the RuuviTag sensor will be employed as a comparative IoT sensor. The RuuviTag's Bluetooth capability offers an alternative method of data collection, allowing for a comprehensive analysis of different wireless technologies in environmental monitoring. This diverse sensor setup will assist in evaluating and comparing the efficacy of RF and IoT sensors in capturing temperature and humidity data.

The system will have controller firmware which will be programmed with unique hexadecimal identifiers for each RF sensor. This strategy is crucial for ensuring distinct and unambiguous communication across the sensor network. By assigning these specific addresses, the implementation of this design aims to effectively segregate data streams from multiple sensors, a vital measure in scenarios where numerous devices might operate on overlapping frequencies. This method significantly reduces the likelihood of data interference and overlap, maintaining the integrity of the communication process within the sensor network.

4. Implementation

4.1. Introduction

Following the completion of a comprehensive literature review and design phase, the implementation detailed in this chapter is as follows. It sets forth the translation of theoretical concepts into a fully functional system that integrates advanced machine learning models, sophisticated data processing techniques, and efficient sensor data management.

Machine Learning Implementation:

The LSTM model's integration is a testament to the application of machine learning in the realm of time-series forecasting. This includes adapting the model architecture to accurately predict temperature trends, essential for the proactive management of pharmaceutical storage conditions.

Data Preprocessing and Time Series Analysis:

The implementation encompasses the use of MySQL Workbench for initial data preprocessing, followed by advanced time series analysis in Google Colab. Techniques from basic data cleansing to complex sequence generation are employed to prepare the dataset for the LSTM model.

Microcontroller and Sensor Integration:

ESP32 microcontrollers and RuuviTag sensors form the hardware foundation for data acquisition. Their configuration and the seamless transfer of data to InfluxDB are highlighted, demonstrating the system's real-time data handling capabilities.

Flask Web Application Development:

The Flask web application stands out as a practical interface between the machine learning backend and end-users. It translates complex predictions into understandable visualizations and actionable alerts.

Alerting and Notification Systems:

The application's backend features a robust alerting mechanism that notifies users via email and SMS when the sensor data deviates from the set thresholds. This proactive alerting system is crucial for maintaining the integrity of the cold chain.

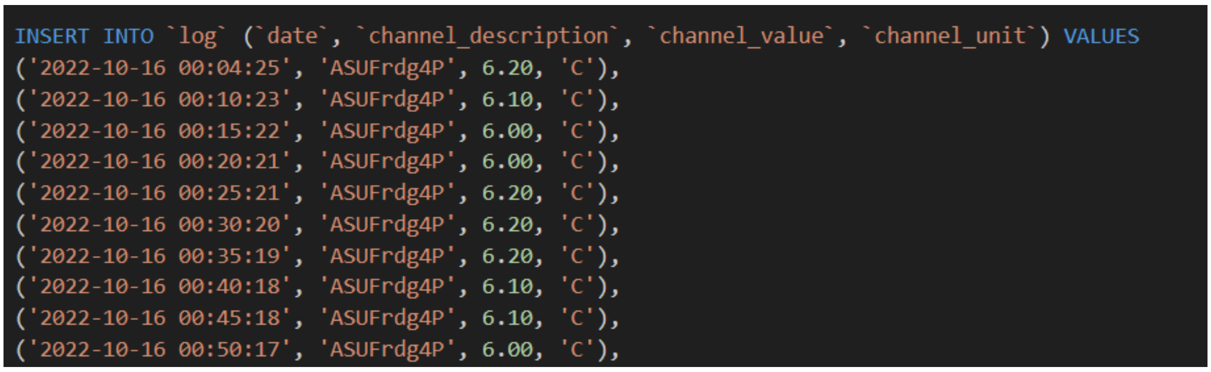
4.2. Data Pre-Processing in MySQL Workbench

4.2.1. Installation and Setup

The initial step in the data pre-processing pipeline for the system was the installation and configuration of MySQL Workbench. This software provides a robust environment for managing the database operations required for our data pipeline. A dummy database was created within MySQL Workbench to serve as the initial receptacle for the raw sensor data scripts.

4.2.2. Data Integration

Each sensor's data, sourced from Kelsius' archive, was exported in the form of MySQL scripts. These scripts represented the raw output of each sensor's monitoring activity. In preparation for analysis, the contents of these scripts were imported into a single, centralized database table referred to as 'log'. This approach to data consolidation was crucial for the subsequent pre-processing stages.



```
INSERT INTO `log` (`date`, `channel_description`, `channel_value`, `channel_unit`) VALUES
('2022-10-16 00:04:25', 'ASUFrdg4P', 6.20, 'C'),
('2022-10-16 00:10:23', 'ASUFrdg4P', 6.10, 'C'),
('2022-10-16 00:15:22', 'ASUFrdg4P', 6.00, 'C'),
('2022-10-16 00:20:21', 'ASUFrdg4P', 6.00, 'C'),
('2022-10-16 00:25:21', 'ASUFrdg4P', 6.20, 'C'),
('2022-10-16 00:30:20', 'ASUFrdg4P', 6.20, 'C'),
('2022-10-16 00:35:19', 'ASUFrdg4P', 6.20, 'C'),
('2022-10-16 00:40:18', 'ASUFrdg4P', 6.10, 'C'),
('2022-10-16 00:45:18', 'ASUFrdg4P', 6.10, 'C'),
('2022-10-16 00:50:17', 'ASUFrdg4P', 6.00, 'C'),
```

Figure 13 – Start of a Sensor MySQL Script Displaying Columns & Rows of Log Table Within Dummy Database

The sensor data, as depicted in the above figure, exhibits several key characteristics essential for understanding and analysing the performance of the cold chain pharmaceutical monitoring system:

Timestamps: Each record includes a precise timestamp under the “date” column, which is necessary for analysis and understanding the sequence of events or conditions. The format of the date column is YYYY-MM-DD HH:MM:SS.

Sensor Identification: The “channel_description” field identifies the sensor or the channel of data collection. In this case, all entries are from a sensor designated as 'ASUFrdg4P'. This identifier is critical for differentiating data streams when multiple sensors are in use.

Measurement Values: The “channel_value” field holds the recorded readings from the sensors. These values are essential for monitoring conditions and determining compliance with the required standards for cold chain management.

Units of Measurement: Each record specifies the unit of measurement (channel_unit), which for these readings is degrees Celsius ('C'). Consistent units are vital for data integrity and avoiding misinterpretation during analysis.

4.2.3. Data Cleansing

The data cleansing phase is a critical step in the data pre-processing. This phase ensures that the dataset used for analysis and machine learning is of high quality by removing any anomalies or irrelevant data that could skew the results.

Three key SQL commands were executed within MySQL Workbench to cleanse the data:

Removing Duplicate Rows: A SQL command was employed to identify and delete duplicate rows from the log table. This was achieved by grouping the data by date and channel_description and retaining only the entry with the minimum id for each group. This ensured that each sensor's readings were unique in time and description, which is essential for accurate analysis.

Eliminating Rows with Null Values: The second command was used to remove any rows that contained null values in either the date, channel_description, or channel_value fields. Null values can indicate missing or corrupt data, which would not be suitable for a robust analysis.

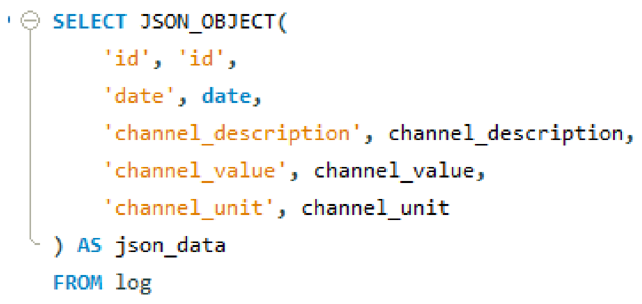
Standardizing Units of Measurement: The final command deleted all rows where the unit of measurement (channel_unit) was not degrees Celsius ('C'). Since the monitoring system is designed to measure temperature in Celsius, any readings in different units were considered irrelevant and thus removed to maintain consistency across the dataset.

These data cleansing operations are essential for maintaining the integrity of the monitoring system's dataset, thereby ensuring the reliability of subsequent analyses and machine learning algorithms applied to the data.

After this was completed, the data is then ready for the next stage in the Pre-Processing pipeline. The next stage is completed within Google-Colab.

4.2.4. Data Export Using MySQL Workbench & JSON

In the subsequent stage of the data pre-processing pipeline conducted within Google Colab, there was a need to transition the dataset format from CSV to JSON. This change was primarily due to the 1,048,576 row limitations inherent in CSV files, which proved inadequate for the dataset's size and complexity. JSON, with its capacity to handle hierarchical data structures more efficiently, offered a more suitable alternative, facilitating seamless data manipulation and analysis in Google Colab. This strategic choice enabled us to bypass the limitations of CSV, ensuring the data was optimally prepared for further processing.



```sql
SELECT JSON_OBJECT(
    'id', 'id',
    'date', date,
    'channel_description', channel_description,
    'channel_value', channel_value,
    'channel_unit', channel_unit
) AS json_data
FROM log
```

Figure 14 – Script for Exporting Data from MySQL Workbench

4.3. Data Pre-Processing in Google Colab

4.3.1 Importing Data from Google Drive

The initial stage of data processing involved importing JSON-formatted data files from Google Drive into the Google Colab environment. This process used the "google.colab" module to mount Google Drive, enabling direct file access within the notebook environment. Subsequently, the JSON file containing pre-processed data was loaded, facilitating further analysis and manipulation in subsequent stages.

4.3.2. Data Integration and Preparation with Pandas

Following the import of JSON files from Google Drive, the next phase involved converting these files into structured data frames for analysis. This conversion was accomplished using Pandas, a powerful and versatile data manipulation library in Python, renowned for its data structure and operations designed to make data cleaning and analysis fast and easy. In this context, two JSON files, resulting from distinct data extraction efforts, were individually loaded into separate Pandas DataFrames. For environments where Pandas is not pre-installed, such as certain local Python setups, it can be installed using the package manager pip, through the command pip install pandas. However, Google Colab environments come with Pandas pre-installed, allowing for immediate use without the need for manual installation.

4.3.3. Categorizing Sensor Data Based on Operational Ranges

The next step was to categorize sensor readings according to predefined temperature ranges. These ranges, provided by Kelsius, represent standard operational ranges for different types of sensors, reflecting their specific use cases in various temperature-controlled environments. As seen in code listing 1 a Python function, categorize_sensor was developed to automatically assign each sensor reading to a category such as 'fridge/cold rooms', 'freezer', 'incubator', 'cryogenic', or 'liquid nitrogen', based on its temperature value. This categorization facilitated targeted analysis across distinct operational contexts, enabling a more nuanced understanding of sensor behaviour and performance within their respective standard operating ranges.

```
# Define the sensor ranges/categories
def categorize_sensor(value):

    if 0 <= value <= 8:
        return 'fridge/cold rooms'
    elif -25 <= value <= -10:
        return 'freezer'
    elif 35 <= value <= 40:
        return 'incubator'
    elif -85 <= value <= -70:
        return 'cryogenic'
    elif -200 <= value <= -190:
        return 'liquid nitrogen'
    else:
        return 'other'


# Apply the categorization function to the 'channel_value'
column
df['category'] = df['channel_value'].apply(categorize_sensor)
```

Code Listing 1 – Categorize Sensors Based on Operational Ranges Provided By Kelsius

4.3.4. Sensor Operational Range Analysis

One of the objectives of code listing 1 was to analyse the distribution and characteristics of sensor data across various operational categories. By leveraging Pandas library, the analysis calculated the total number of records, categorized these records, identified the number of unique sensors within each category, and presented the findings that can be seen in table 3 below.

The results highlighted significant variations in sensor deployment and operational behaviour. Specifically, the distribution of records across all categories alongside a catch-all 'other' category for records falling outside standard ranges. This categorization revealed that a notable portion of the data was classified under 'other', suggesting instances of anomalous behaviour or misclassification, potentially due to sensors being moved or malfunctioning.

Category	Category Records	Category %
Other	4,519,700	45.06%
Fridge/Cold Rooms	4,333,976	43.21%
Incubator	765,315	7.63%
Freezer	296,075	2.95%
Cryogenic	115,158	1.15%

Table 6 – Distribution of Sensor Records Across Pre-Defined Categories

Furthermore, the analysis identified the diversity of sensor usage within each category, emphasizing the specialized nature of certain environments like 'cryogenic' which had a lower count of unique sensors compared to more commonly used settings like 'fridge/cold rooms'.

Knowing the data frame incorporates readings from 50 unique sensors, the below table 4 shows that not all sensors consistently operated within their expected ranges. Specifically, the presence of 44 unique sensors under the 'other' category underscores a significant degree of operational variability or anomalous behaviour. This suggests that a substantial number of sensors deviated from their designated operational environment. This is possibly due to them being moved between different temperature zones, indicating potential malfunctions or other issues.

Category	Unique Sensors
Other	44
Fridge/Cold Rooms	31
Incubator	3
Freezer	7
Cryogenic	1

Table 7 – Analysis That Shows Sensor Records are Falling into More Than 1 Category

4.3.5 Identifying Predominant Operational Categories for Sensors

The following analysis, as detailed in Appendix A4, focused on determining the predominant operational category for each sensor by evaluating the proportion of readings that fell within each specified temperature range. If over 50% of a sensor's readings were confined to a single category, that category was assigned as the sensor's predominant operational environment. Sensors without a clear majority in any category were tagged as 'other', indicating potential anomalies or a lack of specificity in operational deployment.

This approach highlighted key aspects of sensor usage, including the adaptability of sensors across varied temperature conditions and the identification of outliers or misclassified sensors. Specifically, sensors classified under 'other' raised questions about their operational consistency, suggesting they might have been deployed in environments outside their standard operational ranges, or they could be exhibiting erratic behaviour due to technical issues.

sensors exhibiting readings significantly outside their designated operational ranges were excluded, utilizing a threshold beyond two standard deviations. This stringent selection was pivotal for the research work so as to maintain the dataset's integrity, leaving it with 30 sensors whose performance closely adhered to their expected environmental parameters.

The comprehensive review of the data revealed that the majority of excluded sensors were excluded due to their inability to consistently maintain readings within the established temperature ranges set by Kelsius. A subset of sensors was also removed for their deployment in non-specific storeroom environments, characterized by ambient room temperatures rather than the controlled conditions of interest. This action was in alignment with the research hypothesis, which is dedicated to the scrutiny of sensor efficiency and reliability in maintaining the cold chain for pharmaceuticals. Therefore, only sensors that demonstrated strict compliance with the operational ranges provided by Kelsius were retained for analysis.

4.3.6. Addressing Outliers

To ensure the highest level of data accuracy and consistency, an analytical procedure was implemented to identify and address outliers within the remaining sensor dataset. This process was done for refining the dataset further.

Leveraging the capabilities of NumPy, a library utilized for its numerical computations, a function named “replace_outliers” was developed. This function's primary aim was to detect and eliminate outliers by replacing them with NaN (Not a Number) values, thereby mitigating their impact on the dataset's overall integrity. Outliers were defined as readings that deviated more than two standard deviations from the mean of each sensor's data.

This ensured that only values within a reasonable range, as dictated by the calculated mean and standard deviation, were retained. After this cleaning step, the data underwent a time-based interpolation process. This method was chosen to intelligently fill the gaps created by the removal of outliers, ensuring a continuous and coherent dataset ready for further analysis.

Interpolation, particularly time-based, is ideally suited for this scenario as it leverages the temporal sequence of readings to estimate and replace missing or removed values. This method ensures that interpolated values are not only statistically coherent but also contextually relevant, preserving the natural progression and patterns inherent in time-series data. By adopting this approach, the dataset retains its utility for detailed analysis, ensuring that any insights derived are reflective of the true operational conditions.

4.3.7. Time Series Aggregation, Consistent Frequency Setting, and Normalization

This segment focuses on the aggregation of time series data, establishing a consistent frequency across the dataset & normalization of sensor values.

4.3.7.1. Aggregation and Frequency Adjustment

The first step of this segment which is depicted in Appendix A4, involved sorting the dataset by date to preserve chronological order, essential for maintaining the temporal context of sensor readings. To create a unified temporal framework, the analysis implemented a resampling interval of 5 minutes, thereby standardizing the frequency of readings across all sensors. This approach enabled the reassignment of timestamps within each sensor's data, ensuring a continuous and regular sequence of readings without gaps or irregular intervals. By updating timestamps and aligning all sensor data to this uniform temporal structure, the dataset was primed for accurate and synchronized comparison across various operational categories and sensors.

4.3.7.2. Data Normalization

Following the restructuring of the dataset, the normalization of sensor readings was executed using the MinMaxScaler from the scikit-learn library. This transformation adjusted the channel_value readings to a common scale, ranging from 0 to 1, enhancing the comparability of sensor data by mitigating the impact of varying magnitudes and units. Normalization is particularly suitable in this context as it preserves the original distribution of readings while ensuring that the scale differences do not bias the analysis. This step was critical for preparing the dataset for machine learning models and statistical analysis, where uniformity in data scale can significantly influence model performance and insight accuracy.

4.3.8. NaN Values & Interpolation

During the pre-processing phase, the researcher identified the presence of NaN values in the dataset. These Nans were evident just after the dataset's importation into Google Colab. This discovery led to speculation that the removal of null values via MySQL Workbench may have inadvertently left behind non-numeric entries, possibly due to sensor misreads within their operational environments. To ensure the integrity of the analysis, a strategy was adopted to address these NaN values across both the 'channel_value_normalized' and original 'channel_value' columns. Despite employing interpolation to estimate and replace missing values, a subset of 260 NaN values persisted, primarily located at the commencement of datasets for two sensors where adjacent data points for effective interpolation were lacking. As depicted below in code listing 2 backward, and forward fill techniques were applied to these specific instances, leveraging the nearest non-NaN values to ensure data continuity where direct interpolation proved insufficient.

```
# Apply a forward fill or backward fill for edge cases where interpolation
couldn't apply
final_df.fillna(method='ffill', inplace=True)
final_df.fillna(method='bfill', inplace=True)
```

Code Listing 2 – ffill & bfill Method Used for Edge Cases Where Interpolation Couldn't Apply

4.4. Implementation of ARIMA Model

4.4.1. Importing Data from Google Drive and Environment Set-Up

The pre-processed data was imported into a new Google Colab notebook, specifically optimized for statistical modelling and analysis. This step laid the groundwork for implementing and evaluating the ARIMA model on the refined dataset, ensuring a seamless transition from the data pre-processing phase to the intricacies of model training. To facilitate this phase, additional Python libraries were imported to enhance the analysis and visualization capabilities within the notebook. Notably "statsmodels.tsa.arima.model import ARIMA" was used to access the ARIMA modelling functionalities provided by the statsmodels library, a powerful tool for time series analysis. Additionally, import "matplotlib.pyplot as plt" was included to leverage matplotlib, a comprehensive library for creating static, interactive, and animated visualizations in Python. Seen below in code listing 3 are the imports used within this environment.

```
# Imports For Models
import numpy as np
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
```

Code Listing 3 – Current Imports Used Within ARIMA Model Environment

4.4.2. Preliminary Attempts and Model Considerations

Initial attempts to train an ARIMA model on a dataset encompassing all operational categories revealed significant limitations. This approach would fail to account for the distinct behavioural patterns and temperature fluctuations inherent to each category, resulting in a model that inadequately reflected the unique dynamics of each operational environment. Such a generalized model oversimplified the complex, nuanced interactions within the dataset, leading to a poor representation of the actual sensor behaviour across different temperature monitoring scenarios.

Moreover, the ARIMA model faced substantial challenges related to computational demands. The size of the dataset when combining all categories into a single model, coupled with the high-frequency nature of the sensor data, resulted in significant computational strain.

Training the ARIMA model on such a comprehensive dataset proved to be highly resource-intensive, necessitating substantial memory and processing power. Despite utilizing a environment within Google Colab equipped with 60 GB of RAM, the computational limitations were evident through extended run times and frequent crashes. These issues not only hampered the efficiency of the modelling process but also raised concerns about the scalability and practicality of employing ARIMA models for high-volume, high-frequency time series data within this context.

These preliminary modelling attempts underscored the importance of a more nuanced approach to time series analysis in our project. It became clear that a one-size-fits-all model was not feasible given the size and diverse operational environments represented in the dataset.

4.4.3. Category-specific Model Training

Recognizing the need for a more tailored approach, individual models for each category were considered. The decision to start with the "fridge/cold rooms" category was driven by its substantial dataset, offering a foundation for evaluating the ARIMA model's efficacy. The use of “statsmodels.tsa.arima.model.ARIMA” facilitated this process, allowing for nuanced parameter tuning and model evaluation based on the Akaike Information Criterion (AIC). AIC is a measure used in statistics to compare the relative quality of statistical models for a given set of data. It rewards model accuracy while penalizing excessive complexity, helping to identify the model that best explains the data with the least number of parameters. This criterion was helpful in selecting the most suitable ARIMA model configuration for each specific category, ensuring an optimal balance between model complexity and predictive performance.

4.4.4. Challenges with Parameter Selection and Model Complexity

In the pursuit of optimizing ARIMA models for the dataset, the research initially employed Auto ARIMA, aiming to automate the process of identifying the most efficient model parameters by exploring various combinations. This method was expected to streamline the model selection process, offering an efficient pathway to determining the optimal configuration for the models based on AIC & RMSE values. However, the practical application of Auto ARIMA in this context quickly uncovered significant computational challenges. The method faced severe limitations almost immediately upon execution it became apparent that the computational intensity of Auto ARIMA, coupled with the complexity and volume of the dataset, led to repeated system crashes. These crashes occurred after the evaluation of merely one or two parameter combinations, primarily due to memory exhaustion, despite having 60 GB of RAM available. This scenario underscored the prohibitive computational demands associated with using Auto ARIMA & ARIMA, given the dataset's characteristics.

Confronted with the impracticality of Auto ARIMA, owing to these memory constraints, the researcher was compelled to reassess strategy for model optimization. This reassessment led to the adoption of predefined parameter sets as an alternative approach as seen in Appendix A5. By manually specifying and testing various parameter combinations, the researcher sought to strike a balance between model complexity and computational efficiency in a more manageable and resource-conscious manner. This shift from automated to manual parameter selection allowed for a more incremental exploration of the parameter space, albeit at the cost of increased hands-on effort and time. Through selection and evaluation of each parameter set, the aim was to uncover a configuration that achieved an acceptable balance between accuracy and computational demand.

This phase of the research highlighted the critical challenge of balancing the complexity of statistical models with the limitations of computational resources, especially in the context of extensive time series analysis.

4.4.5. Model Suitability and Data Frequency Considerations

It became apparent that the ARIMA model, traditionally suited for lower-frequency time series data, struggled with the high-frequency nature of the sensor data. The 5-minute interval data posed significant challenges for ARIMA, exacerbating issues of computational efficiency and model stability. Moreover, the time series length and the artificial timeline introduced by the aggregation method used previously added layers of complexity. The process of aggregating sensor data into a consistent 5-minute interval effectively created an extensive, continuous time series, significantly elongating the dataset. This not only increased the computational load but also introduced potential distortions in the temporal dynamics of the data, further complicating the modelling process.

4.4.6. Concluding Remarks and Future Directions

The initial application of ARIMA modelling to the "fridge/cold rooms" data segment provided critical insights into the inherent limitations and nuanced considerations essential for effective time series analysis within the realm of cold chain monitoring. This phase of the project highlighted the significant challenges introduced by the dataset's high-frequency nature and substantial volume, compounded by the computational difficulties encountered during model training. These challenges, alongside the ARIMA model's limitations in accommodating the dataset's granularity and the extended timeline resulting from prior aggregation methods, necessitate a shift towards alternative modelling solutions. Future research will thus focus on exploring more adaptable, efficient, and scalable modelling approaches that can better address the dataset's complexity, facilitating a nuanced understanding and management of temperature variations across different operational sensor categories.

Building on the foundational insights gained from applying ARIMA models, the project will next explore the use of Long Short-Term Memory (LSTM) models. This decision is informed by the previously discussed capabilities of LSTM in the literature review, highlighting its suitability for handling the dataset's characteristics. The research will focus on implementing LSTM to address the challenges identified with ARIMA, particularly its limitations with high-frequency, large-volume datasets. The aim is to leverage LSTM's strengths to enhance predictive accuracy and computational efficiency.

4.5. Implementation of LSTM Model

4.5.1. Importing Data and Environment Set-Up

Transitioning from the exploration of ARIMA models, the research progresses to implementing Long Short-Term Memory models to address the identified challenges with high-frequency, voluminous datasets. This phase initiates with importing the pre-processed data into a newly optimized Google Colab notebook, tailored for deep learning applications, including LSTM model development. To support this endeavour Key imports shown in Appendix A6 include TensorFlow for building and training LSTM models, Keras for model architecture and hyperparameter tuning, and returning libraries for data manipulation and model evaluation.

4.5.3. Model Architecture and Hyperparameter Tuning

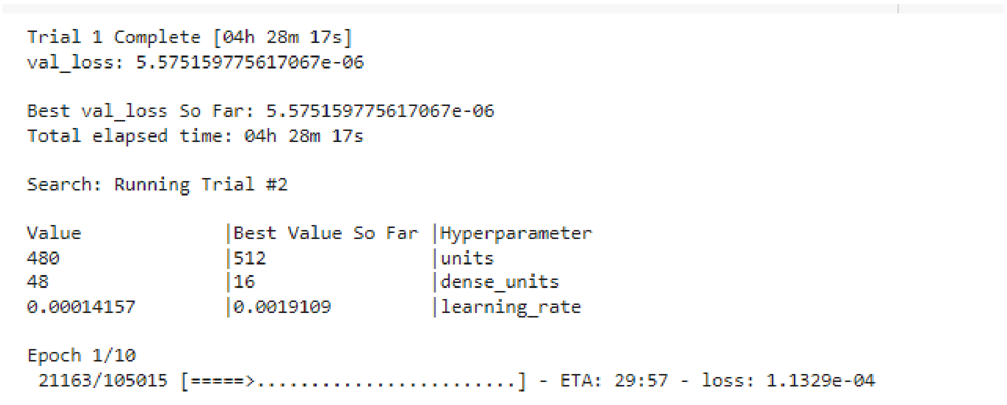
This segment of the research investigates the steps of designing the LSTM model's architecture and hyperparameter tuning, essential for enhancing the model's performance and accuracy. Leveraging TensorFlow and Keras, the architecture is crafted, incorporating multiple LSTM layers to capture the temporal dependencies and patterns within the sensor data effectively. As shown below in Appendix A7 the process of hyperparameter tuning employs Keras Tuner, aiming to identify the optimal configuration that strikes a balance between the model's complexity and its predictiveness. This approach ensures the LSTM model is tailored to the dataset's characteristics, efficiency, and accuracy.

4.5.4. Hyperparameter Optimization

The process detailed below in Appendix A8 embarks on applying the LSTM model to various categories within the dataset. Engaging in trial runs for model training, optimization, and evaluation. Through these steps, the model is subjected to a series of hyperparameter tuning sessions using Keras Tuner, and its performance is meticulously assessed across different segments of the dataset. This systematic approach facilitates the identification of the most effective model configuration for each category, ensuring the LSTM model achieves the highest predictive accuracy possible.

Utilizing Keras Tuner, the research delves into optimizing the model's architecture defined in Appendix A8 through an empirical process. As illustrated in figure 16, this involves conducting trials with different hyperparameters to determine the most effective configuration that minimizes loss and enhances the model’s predictive accuracy.

Figure 15 – Beginning of Trial Run 2 in Search of Best Hyperparameters for the Dataset



```
Trial 1 Complete [04h 28m 17s]
val_loss: 5.575159775617067e-06

Best val_loss So Far: 5.575159775617067e-06
Total elapsed time: 04h 28m 17s

Search: Running Trial #2

Value             |Best Value So Far |Hyperparameter
480               |512               |units
48                |16                |dense_units
0.00014157        |0.0019109         |learning_rate

Epoch 1/10
 21163/105015 [=====>........................] - ETA: 29:57 - loss: 1.1329e-04
```

Throughout the training process, checkpointing mechanisms are employed to save iterations of the model that show improvement, ensuring that the best-performing model is retained. Upon concluding the training and hyperparameter optimization, the best model is evaluated on a test set to assess its predictive performance, typically using metrics like RMSE (Root Mean Squared Error) to quantify accuracy.

4.5.5. Splitting & Sequencing of Data

After identifying and saving the best LSTM model configuration, the next step involved transferring this model to a separate Google Colab notebook specifically arranged for a model training regime.

In this environment, the methodology for preparing the dataset adopted an approach that mirrors real-world conditions more closely. Recognizing the diversity and distinct characteristics inherent in sensor data, the decision was made to separate the sensors into two groups: one for training and one exclusively for testing. This choice reflects an intent to model a scenario where the LSTM model is trained on data from a set of sensors and then tasked with making predictions on data from a completely new sensor, mimicking a realistic application where the model must generalize across different operational contexts. This methodology is detailed in Appendix A9.

With the dataset divided, attention then shifted to task of data sequencing. This technique involves organizing historical data into sequences that serve both as input for the model and as a basis for predictions about future states. Given the temporal nature of the dataset, this approach is vital for capturing the inherent patterns and dynamics over time, which are crucial for accurate forecasting.

As depicted in Appendix A9 the sequencing process was carried out by generating ordered sets of data points, each set comprising 144 consecutive readings to predict the subsequent values. This length was chosen to provide a comprehensive view of the past 12 hours, offering the LSTM model a detailed temporal context for making predictions. By employing sequences of this length, the model is better positioned to identify and learn from the temporal correlations within the data, enhancing its ability to forecast future events with higher precision.

This careful preparation of the data underscores the research's commitment to developing a LSTM model not just with high predictive accuracy, but also with strong generalization capabilities. It reflects an understanding that for machine learning models to be truly effective in real-world applications, they must be adaptable, able to handle new, unseen data with the same level of accuracy as the data they were trained on.

4.5.6. Training of First LSTM Model

As detailed in Appendix A10, the LSTM model is set up for its 1st full training. Initially, the model, undergoes compilation integrating the Adam optimizer and MSE as its loss function. This setup is pivotal for refining the model's weights through iterative learning from the training data, aiming to minimize prediction errors in a continuous feedback loop.

Following compilation, the model proceeds to the training phase, where it is fed batches of pre-sequenced sensor data. Each epoch represents a full pass through the training dataset, allowing the model to adjust its internal parameters based on the learned temporal patterns and the associated errors. The inclusion of validation data offers a crucial checkpoint, enabling the assessment of the model's performance on unseen data after each epoch, which is instrumental in gauging its generalization capability.

The training process is further enhanced by callbacks for early stopping and model checkpointing. Early stopping safeguards the model against overfitting by halting training if the validation loss fails to improve over a set number of epochs. Concurrently, model checkpointing preserves the state of the model at its peak performance on the validation set, ensuring that the best version is retained and available for subsequent evaluation and deployment.

Upon completion, the trained model is poised for evaluation, where its predictive accuracy is scrutinized using the test dataset. This critical step quantifies the model's ability to generalize its learned patterns to new, unseen data, embodying the ultimate test of its predictive prowess. The outcomes of this evaluation phase are instrumental in informing further refinements to the model or signifying its readiness for real-world application scenarios.

4.5.7. Model Evaluation

Upon initial evaluation, it was determined that the first iteration of the LSTM model, as configured in Appendix A10, was not ideally suited for practical applications. Specifically, the model was designed to predict only a single future value (5 minutes ahead) per input sequence. This limited forecasting capability proved to be counterintuitive for scenarios requiring a broader temporal insight, as it restricted the application's utility to very short-term predictions.

To enhance the model's utility and align it with practical needs, the LSTM architecture was subsequently modified to predict multiple future steps. The revised model is now capable of forecasting up to 12 steps ahead, equating to a full hour of future data. This adjustment not only extends the predictive horizon but also significantly improves the model's applicability in real-world scenarios, where decisions often rely on understanding trends and changes over longer periods.

The enhancements to the LSTM model were built upon the foundational tuning and parameters established during the first model's development. This iterative refinement as seen in Appendix A11 process involved leveraging the insights gained from the initial model's performance, particularly in terms of learning dynamics and error minimization strategies. By adjusting the model architecture to predict multiple steps, the training parameters including the number and size of LSTM layers, the batch size, and the learning rate were optimized to better capture and predict the temporal patterns over a longer horizon.

This progressive approach not only preserved the strengths of the initial model but also addressed its limitations, enabling a more robust and versatile prediction tool. The subsequent training sessions incorporated refined callbacks and optimization strategies, ensuring that the model not only learns efficiently but also generalizes well to new data, thereby improving its practical efficacy for deployment in dynamic environments.

4.5.8. Model Exportation

Following the successful enhancement and validation of the LSTM model, the next critical step was to export and store the model for future use within the application. Utilizing TensorFlow's built-in capabilities, the model was serialized and saved in the TensorFlow SavedModel format, which encapsulates the complete model architecture, weights, and computation graph.

4.5.9. Focusing on the "Fridge/Cold Room" Category

The substantial computational resources required, and the considerable size of the dataset necessitated an approach in this research. Consequently, the scope was narrowed to develop predictive models specifically the "Fridge/Cold Room" category. This focused category allowed for a more manageable and in-depth exploration of predictive accuracy in a critical application area.

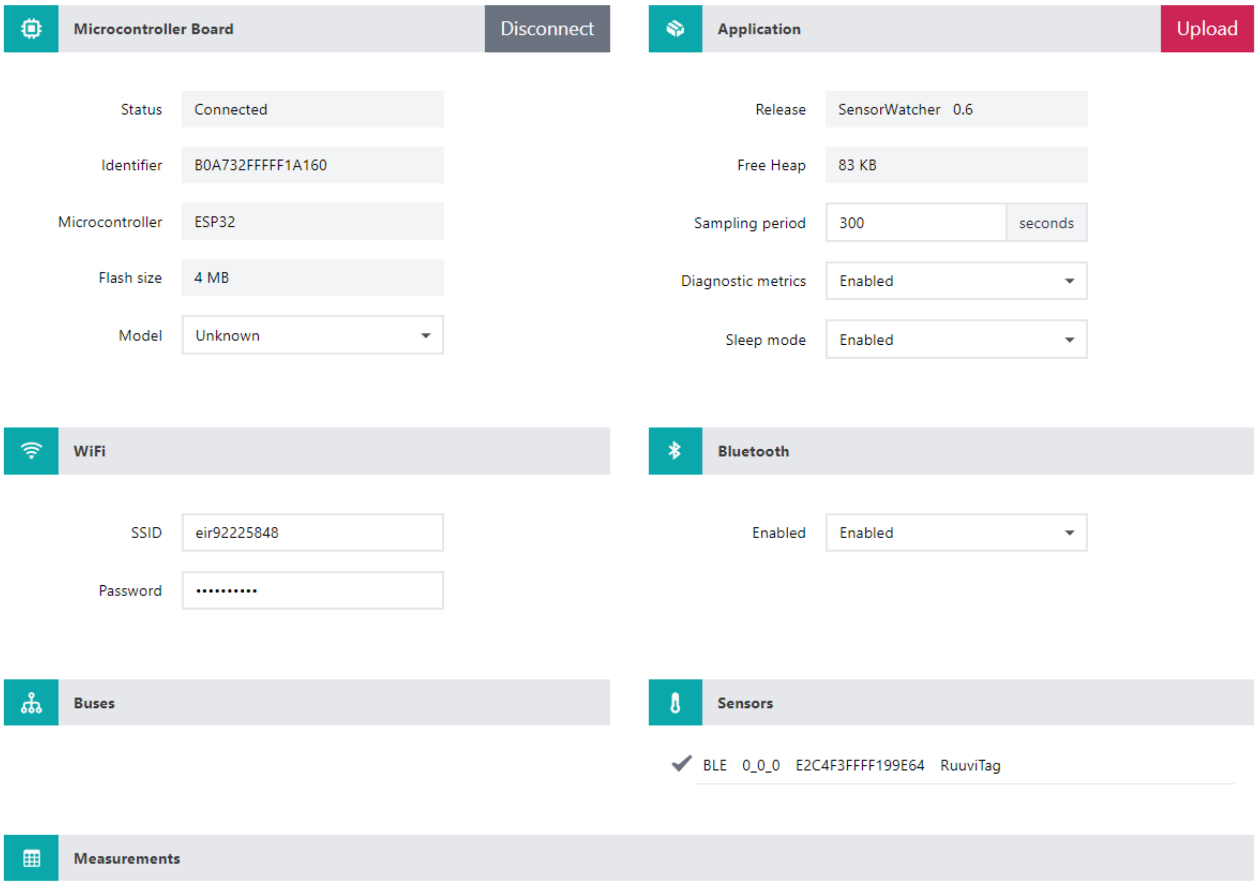
By concentrating exclusively on the "Fridge/Cold Room" category, the research efforts were optimized within the practical limits of available computational resources.

This strategic limitation not only ensured the feasibility of the project but also enhanced its effectiveness by allowing for a thorough examination of specific dynamics and requirements typical to refrigeration units. This focus highlights the importance of well-defined project scopes in managing and executing data-intensive research within the constraints of available resources.

4.6. Micro-Controller, Sensor & influx DB

During the research, an ESP32 microcontroller was utilized, chosen for its reliability and versatility in IoT applications. This microcontroller was configured to work in tandem with a RuuviTag, an open-source environmental sensor. The RuuviTag is known for its robustness provision of accurate data and open-source library.

A critical aspect of the setup involved the automatic configuration of the ESP32 and RuuviTag to effectively communicate and log data to InfluxDB, a time-series database optimized for high-availability storage and retrieval of time-stamped data. To streamline this process, the researcher employed a web-based setup tool available at https://movoki.com/setup/. This



Status	Connected
Identifier	B0A732FFFFF1A160
Microcontroller	ESP32
Flash size	4 MB
Model	Unknown
Release	SensorWatcher 0.6
Free Heap	83 KB
Sampling period	300 seconds
Diagnostic metrics	Enabled
Sleep mode	Enabled
SSID	eir92225848
Password	••••••••••
Enabled	Enabled

✓ BLE 0_0_0 E2C4F3FFFF199E64 RuuviTag

Figure 16 – Movoki Web Application that Automatically Configures ESP32 with Sensor,InfluxDB & Wi-Fi Configuration

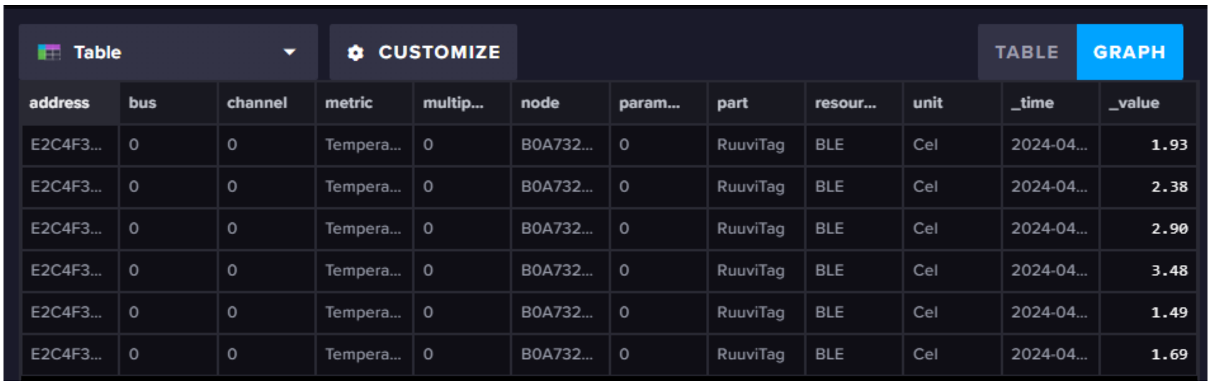
tool facilitated a user-friendly interface for the configuration of the microcontroller, as depicted in figure 15 below.

The interface allowed for the effortless adjustment of settings corresponding to the ESP32 and RuuviTag, such as Wi-Fi and Bluetooth connectivity and backend integration with

InfluxDB. It provided a straightforward means of customizing various parameters without the need for in-depth programming, thereby saving valuable time and reducing complexity.

The setup tool proved to be particularly advantageous in managing the measurements and logging parameters that are critical in a research environment where data integrity and accuracy are of utmost importance. It allowed the researcher to focus on the data analysis rather than the intricacies of hardware and software configuration.

By leveraging this automatic configuration tool, the research project was able to ensure that data collected from the "Fridge/Cold Room" category was consistent, reliable, and appropriately channelled into the InfluxDB bucket depicted in figure 17 for real-time monitoring and subsequent analysis. This efficiency was instrumental in managing the extensive datasets required for the LSTM model's training and evaluation, aligning with the project's focus on computational resource optimization and specific category insights.



address	bus	channel	metric	multip...	node	param...	part	resour...	unit	_time	_value
E2C4F3...	0	0	Tempera...	0	B0A732...	0	RuuviTag	BLE	Cel	2024-04...	1.93
E2C4F3...	0	0	Tempera...	0	B0A732...	0	RuuviTag	BLE	Cel	2024-04...	2.38
E2C4F3...	0	0	Tempera...	0	B0A732...	0	RuuviTag	BLE	Cel	2024-04...	2.90
E2C4F3...	0	0	Tempera...	0	B0A732...	0	RuuviTag	BLE	Cel	2024-04...	3.48
E2C4F3...	0	0	Tempera...	0	B0A732...	0	RuuviTag	BLE	Cel	2024-04...	1.49
E2C4F3...	0	0	Tempera...	0	B0A732...	0	RuuviTag	BLE	Cel	2024-04...	1.69

Figure 17 – Table that Shows Temperature Records within the influxDB Bucket located in EU Central

4.7. Flask Web Application

The development of a Flask web application forms an integral part of this research, serving as the user interface for real-time data visualization and predictive analysis. This web application bridges the gap between raw sensor data and actionable insights, offering a user-friendly platform for monitoring refrigeration units, specifically in the "Fridge/Cold Room" category.

4.7.1. Application Architecture and Setup

The application is architected to function as a central hub for data ingestion, processing, and visualization. At the onset, the App.py file is structured with meticulous attention to the imports and configurations necessary for a smooth operation. A notable environment configuration ensures that TensorFlow operations are designated to run on CPU to adapt to environments where GPU resources are not viable or are otherwise dedicated to other tasks. This strategic setup underscores the application's design for adaptability across diverse operating environments, prioritizing accessibility, and ease of deployment.

4.7.2. InfluxDB's Architecture and Flask Backend Integration

The decision to integrate InfluxDB into the Flask application's architecture was driven by its specific optimization for handling time-series data. InfluxDB excels where data is not only voluminous but also inherently time-stamped, as is typical with sensor data in cold chain monitoring. This capability allows InfluxDB to perform exceedingly well under the high write loads and query demands characteristic of real-time monitoring systems. Time-series databases like InfluxDB are adept at managing 'time' as a primary index, which is fundamental for the temporal analysis and predictive modelling undertaken in this project. The actual query used to retrieve data from InfluxDB, highlighting the practical implementation of these concepts, is detailed in Appendix 13.

4.7.3. Implementation of the LSTM Model in the Flask Application

The integration of the LSTM model into the Flask application is facilitated through the "PredictionModel" class, a vital component that abstracts the intricacies of machine learning workflows. This class plays a crucial role in interfacing the raw sensor data with the LSTM's advanced predictive functions. The class's design reflects a high degree of modularity, enabling it to manage the LSTM model, which resides at a specified path within the application's directory structure. The path acts as a pointer, directing the application to the model's storage location. This architecture choice provides the adaptability needed to accommodate updates to the model, including retraining, saving, and replacing the LSTM model without disrupting the core functionality of the Flask application.

In practice, when the PredictionModel class is instantiated, it assumes the responsibility of conditioning the sensor data for the prediction process. This preparation entails normalizing the data to match the scale used during model training a step to ensure the predictions' accuracy. Additionally, it shapes the incoming data into sequences that conform to the LSTM's input requirements, ensuring that the model's internal structure is properly fed with appropriately structured data. These sequences are pivotal for the LSTM to capture and learn from the temporal dependencies present in the sensor data.

The PredictionModel class offers a suite of methods integral to the prediction process:

Normalization: Implementing data scaling to ensure that input features are on the same scale as the training data.

Sequence Generation: Converting raw time-series data from influxDB into a structured format suitable for LSTM processing. Each sequence being 24 hours long.

Model Loading: Instantiating the LSTM model from the given file path, encapsulating the complexities of TensorFlow model deployment.

Prediction: Running the model prediction using processed data and extracting the forecasted temperature values.

In Figure 18 on the next page, we observe the output of the prediction model in the web application's temperature dashboard. The graph displays two distinct lines: one representing the actual recorded temperature data and the other depicting the predicted temperatures. The actual data is shown as a solid blue line, indicating the temperature readings captured over time. Following the last recorded data point, the prediction model's output is visualized as a dashed red line, providing a forecast of future temperatures. This predictive feature is instrumental for proactive temperature management within the monitored environment. For a comprehensive examination of the underlying code and logic, the complete PredictionModel class is documented in Appendix 12.

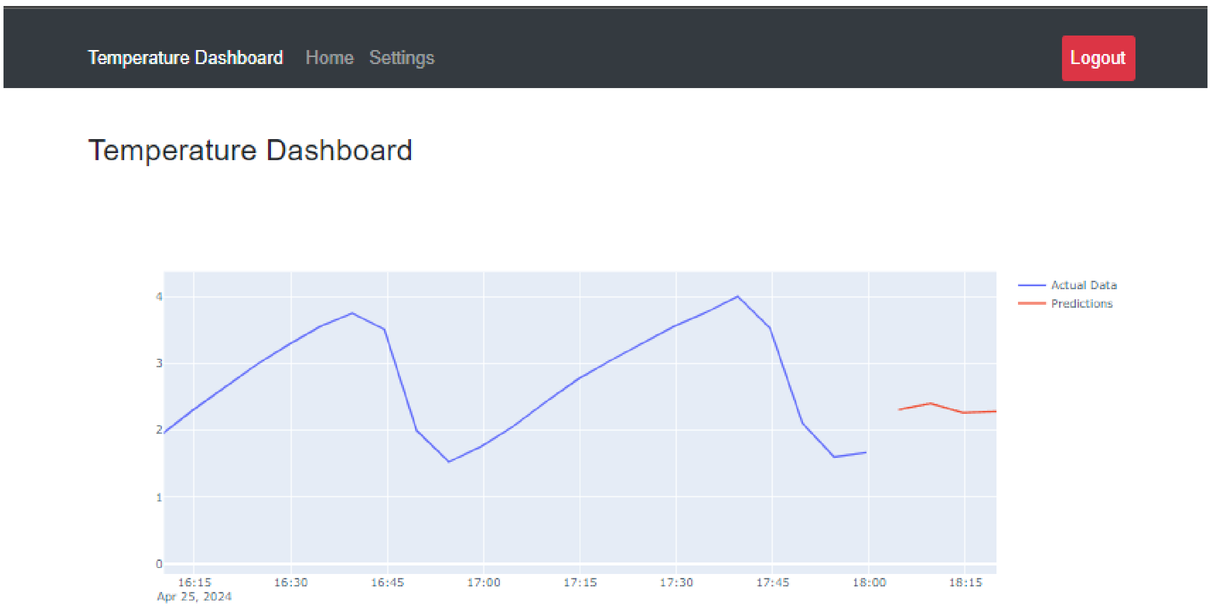


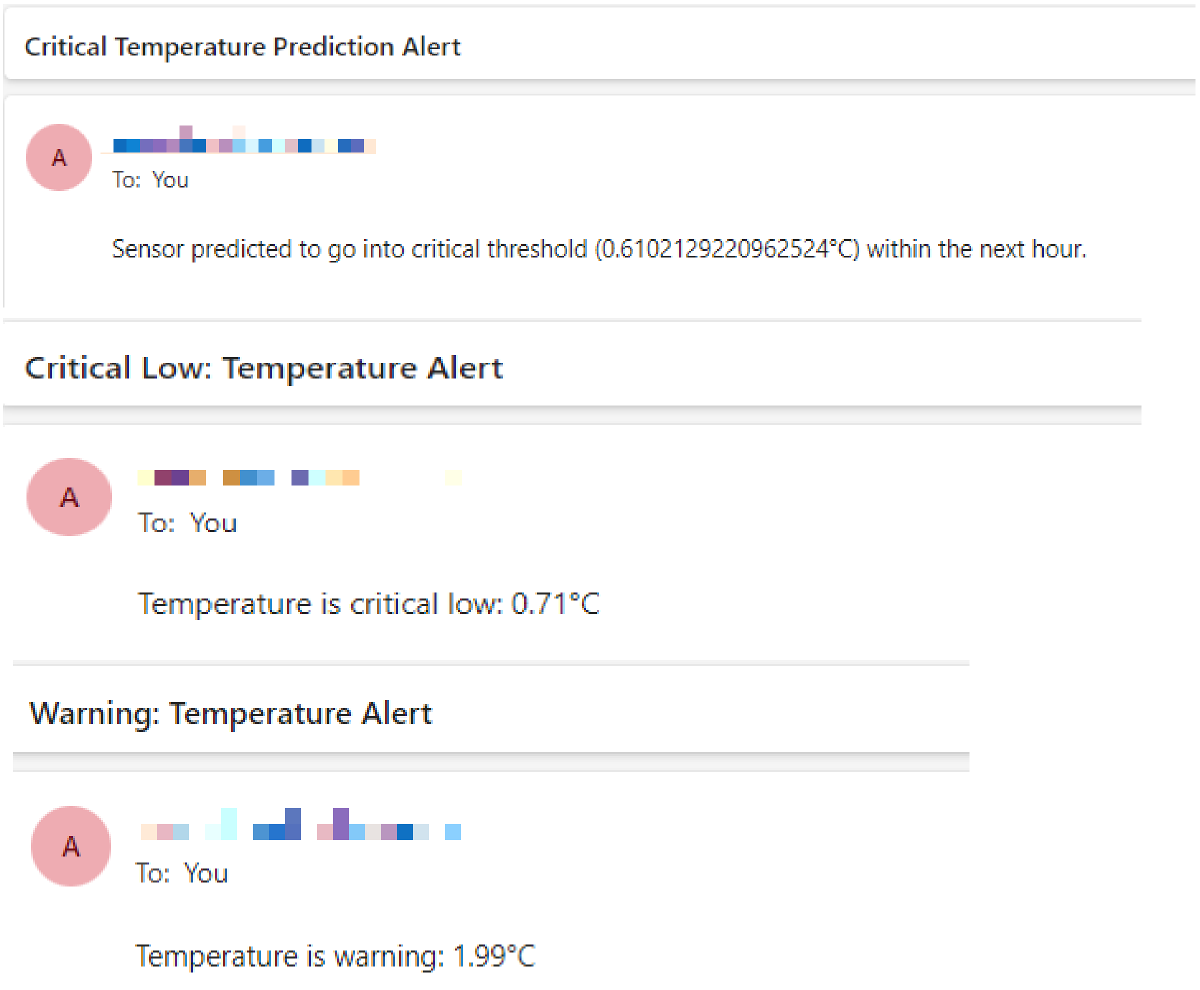
Figure 18 – Prediction Model Visualization demonstrates the model's capacity to forecast future temperature trends based on historical data

4.7.4. Alerting Functionality

The Flask web application's alerting framework plays a crucial role in the maintenance and operational management of refrigeration units by providing timely notifications when temperature readings deviate from predefined thresholds. This framework is implemented through a series of well-defined functions within the Alert.py module, leveraging external communication services to ensure effective alert dissemination.

4.7.4.1. Implementation of Email and SMS Notifications

The framework utilizes two primary channels for sending alerts: email via SendGrid and SMS via Twilio. This multi-channel approach ensures that notifications are received even if one service encounters an issue. The send_email function initializes the SendGrid API client with credentials securely fetched from environment variables. It constructs the email with the required attributes sender, recipient, subject, and content before dispatching it through the SendGrid service. Similarly, the send_sms function uses the Twilio client, configured with credentials from environment variables, to send text messages directly to the user's phone. See



Critical Temperature Prediction Alert

A [redacted]

To: You

Sensor predicted to go into critical threshold (0.6102129220962524°C) within the next hour.

Critical Low: Temperature Alert

A [redacted]

To: You

Temperature is critical low: 0.71°C

Warning: Temperature Alert

A [redacted]

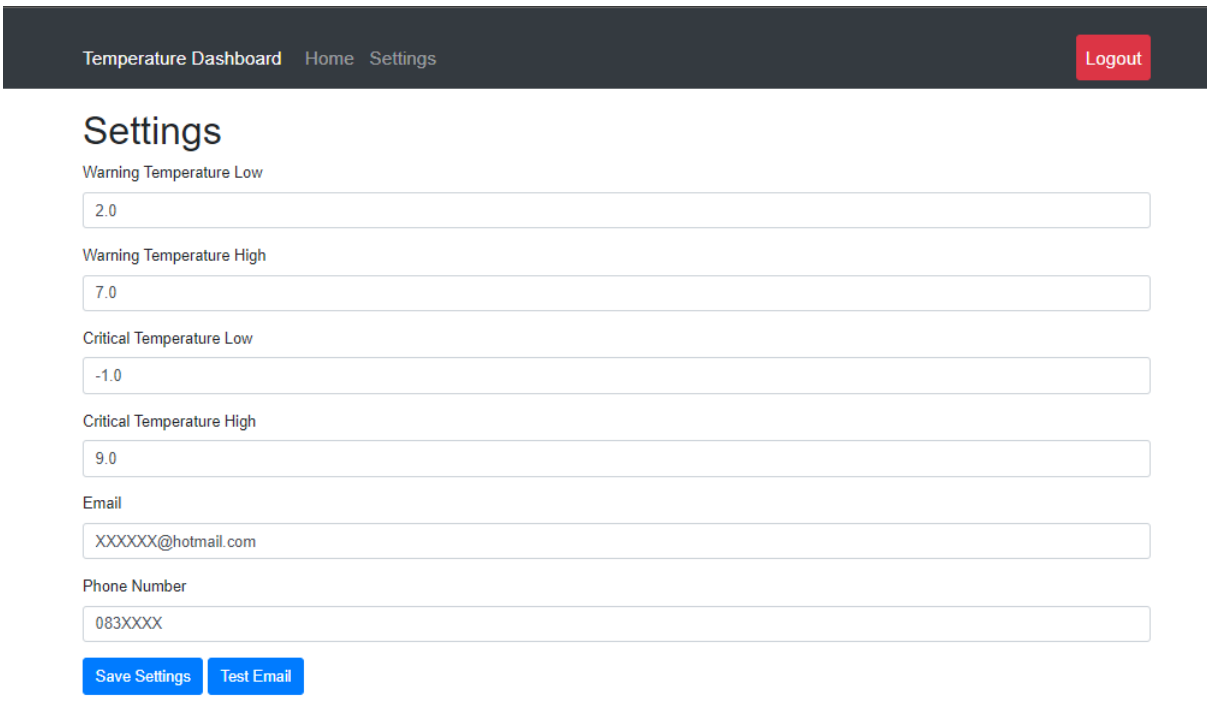
To: You

Temperature is warning: 1.99°C

Figure 19 – Sample of all Alert Types That the User Can Receive

4.7.4.2. Configuration of Alert Thresholds and Conditions

The core of the alert system is the dynamic configuration capability, which allows users to set specific temperature thresholds for warnings and critical alerts via the Settings route depicted below in figure 19. These thresholds are stored in a configuration file (Config.json), which the load_config and save_config functions manage. The application reads these thresholds to determine when to trigger alerts, assessing sensor data against these user-defined parameters.



Settings

Warning Temperature Low

2.0

Warning Temperature High

7.0

Critical Temperature Low

-1.0

Critical Temperature High

9.0

Email

XXXXXX@hotmail.com

Phone Number

083XXXX

Figure 20 – Settings Page on Web Application That Allows Thresholds to be Configured & Saved to a Config File

4.7.4.3. Alert Timing and Frequency Control

Alert fatigue is mitigated by the should_send_alert function, which checks the elapsed time since the last alert was sent. By maintaining a record of the last alert time in the configuration file, the system ensures that alerts are not sent too frequently, which could desensitize users to potential emergencies. This function compares the current time against the last alert time and respects a minimum interval before sending another alert, thus balancing responsiveness with prudence.

4.7.4.4. Proactive Alerting Based on Predictions

Proactive alerting is achieved through the alert_for_predicted_values function. This function analyses predicted future temperature values, provided by the LSTM model, to forecast potential threshold breaches before they occur. If the predicted temperatures cross the configured thresholds, the system proactively notifies the user. This early warning system is particularly valuable in preventing damage or loss by allowing pre-emptive measures to be taken.

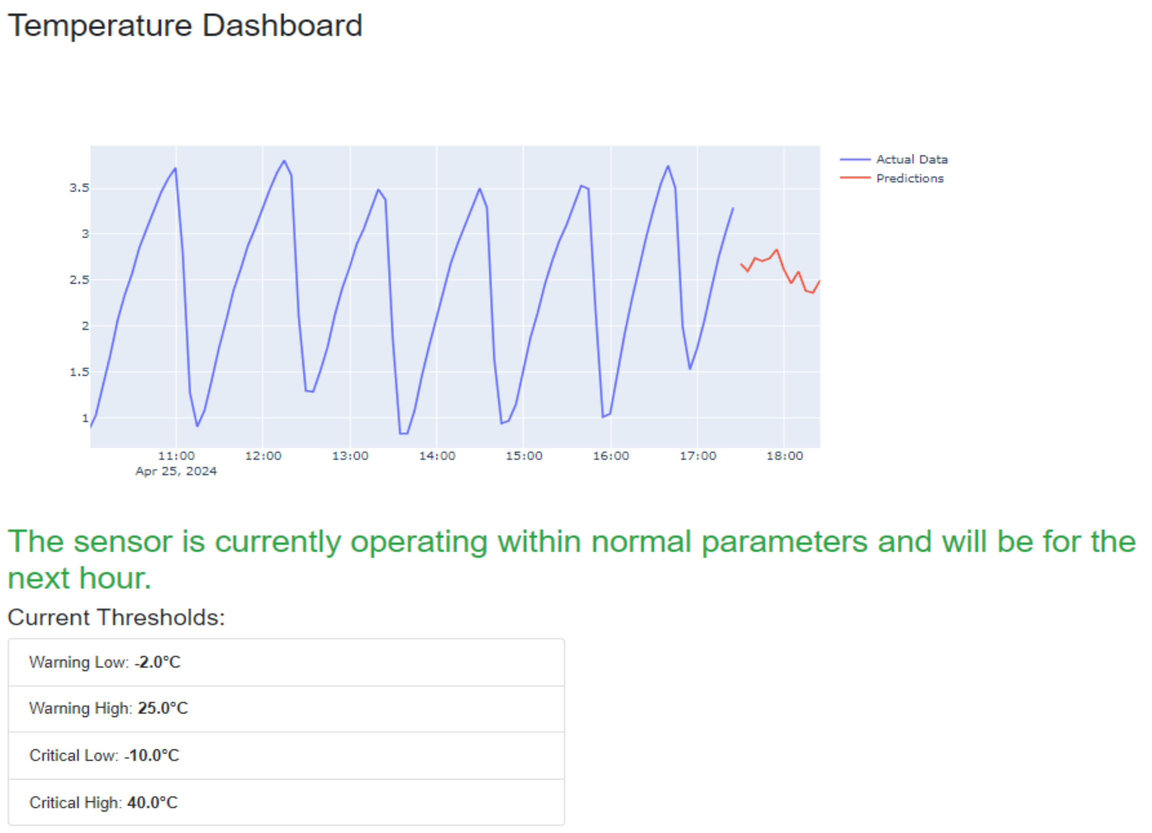
4.7.5. Status Messages for Sensor Monitoring

The check_current_status function is called to determine the immediate status of the sensor readings based on the latest data. It evaluates whether the temperature is within the normal range or if it has entered a warning or critical state. The evaluate_sensor_status function goes a step further by incorporating predictions into the status assessment, giving a comprehensive overview that includes past, present, and future data. This function uses both current and predicted values to assess the overall system status, enhancing the decision-making process with a broader data context. The possible statuses are as follows:

4.7.5.1. Normal

"The sensor is currently operating within normal parameters and will be for the next hour."

This status message as illustrated in figure 20 indicates that the sensor's current readings and predicted values for the next hour are within the safe operational thresholds. It assures that the system is functioning optimally and does not require immediate attention.



Temperature Dashboard

The sensor is currently operating within normal parameters and will be for the next hour.

Current Thresholds:

Warning Low: **-2.0°C**
Warning High: **25.0°C**
Critical Low: **-10.0°C**
Critical High: **40.0°C**

Figure 21 – System Functioning Optimally within "Normal" Ranges and Does Not Require Immediate Attention Displaying on Main Dashboard

4.7.5.2. Warning

Attention: Please check the sensor as it is in a Warning Threshold.

This alert is triggered when the sensor readings approach a pre-set warning threshold, suggesting that conditions are veering towards an out-of-norm state. It serves as a precaution to prompt preliminary checks or adjustments before a potential escalation to critical levels.

4.7.5.3. Critical

Warning: The sensor has reached a critical threshold!

This message is issued when the sensor readings have reached or exceeded critical thresholds. It signals a possible urgent issue requiring immediate action to prevent system damage or failure, highlighting the severity of the situation.

4.7.5.4. Default

Status currently unavailable.

This status is used when the sensor data is inconclusive, or the system is unable to retrieve or calculate the current status. It indicates that there may be a technical issue with data transmission or processing, suggesting that verification of the system's operational integrity is necessary.

4.7.5.5. Proactive

Be proactive: The sensor is predicted to reach a threshold within the next hour.

This forward-looking alert is generated based on predictive data analysis, which forecasts that the sensor will likely reach a threshold level within the next hour. It encourages proactive measures to adjust the system or prepare for potential impacts, emphasizing the predictive capabilities of the monitoring system.

These status messages play a crucial role in the sensor monitoring application, providing clear and immediate communication regarding the state of the refrigeration units. They enable users to maintain efficient operational oversight and ensure timely responses to changing conditions.

5. Testing Strategy

5.1. Introduction

This chapter delves into the testing methodologies adopted to validate the research findings, specifically focusing on both Functional and Non-Functional Requirements as detailed in Chapter 3.3 titled Application Requirements. The chapter aims to establish a testing framework, utilizing a blend of Black-Box and White-Box testing strategies to ensure comprehensive coverage and validation of the system's functionalities and performance attributes.

5.2. Black-Box Testing

Black box testing is a method of software testing where the tester does not have knowledge of the internal mechanisms or coding framework of the system under test. This implies that the tester's attention is exclusively on the external functionality of the software, without any insight into its internal code structure. The term “black box” is derived from the concept that the internal operations of the system remain concealed or “enclosed” from the perspective of the tester (Das 2023), it includes:

Integration Testing: Tests the interactions between different modules or components of a system to ensure they work together as expected.

Security Testing: Assesses the software's ability to protect data and resist attacks, ensuring that all security measures are robust and effective.

System Testing: Involves testing the complete system to verify that it complies with the specified requirements.

User Interface (UI) Testing: Focuses on the graphical user interface of the application, ensuring it is user-friendly, responsive, and functions as intended.

Comparative Testing: Involves comparing different components or systems against each other to evaluate their relative performance.

Functional Testing: Tests the software's functions to ensure they operate in line with the requirement specifications.

Scalability Testing: Assesses the software’s capacity to handle growth, such as an increasing number of users or data volume.

Usability Testing: Evaluates the software’s ease of use, including its user interface and overall user experience.

Compliance Testing: Checks if the software adheres to laws, regulations, standards, or guidelines, particularly in terms of data privacy and security.

5.3. White-Box Testing

The counterpart to Blackbox testing, White Box Testing is a method where the internal composition, architecture, and coding of software are known. This approach is aimed at checking the flow from input to output and enhancing aspects such as design, usability, and security. In this technique, testers have access to the code, which is why it's also known as Clear Box Testing, Open Box Testing, Transparent Box Testing, Code-Based Testing, or Glass Box Testing (Hamilton 2023). Some testing strategies for white box testing are:

Integration Testing (White Box Context): Analyses the data flow and control flow within the integrated modules, focusing on internal workings.

Validation Testing: Verifies that the software meets all requirements and expectations, particularly focusing on the correctness of the implemented logic.

Security Testing (White Box Context): Involves in-depth testing of the internal security features, such as code-level security protocols and encryption mechanisms.

Efficiency Testing: Examines the software's effective use of system resources, like CPU, memory, and network, and its overall performance efficiency.

5.4. Testing Strategy

The tables 8 and 9 depicted below provide a structured overview of the testing strategies for both Functional and Non-Functional Requirements of the proposed system.

Table 8 – Functional Requirements Testing Strategy

5.4.1. Functional Requirements

Test ID	Description	Testing Strategy	Testing Category	Details of Testing Strategy	Result
FR1	The system must be able to collect temperature data from wireless sensors connected to the microcontroller.	Integration Testing	Black Box	Test integration of sensors with the microcontroller for accurate data collection.	Successful data collection from sensors.
FR2	The system should transmit data securely from the microcontroller to a centralized server or cloud-based system for further analysis.	Security Testing	Black Box	Validate the security of data transmission, ensuring encryption and secure channel use.	Data transmission from ESP32 to influxDB Cloud.
FR3	The system must integrate machine learning algorithms for anomaly detection and time-series forecasting of temperature data.	Integration Testing	White Box	Verify the correct integration of ML algorithms within the system's architecture.	LSTM algorithm integrated and functioning as expected.

FR4	The system must continuously monitor temperature data in real-time, ensuring immediate detection of any deviations from the set thresholds. It should automatically log all temperature readings with timestamps.	System Testing	Black Box	Assess the system's ability to continuously monitor and log temperature data accurately.	Real-time monitoring verified; all deviations detected promptly.
FR5	The application must have a web-based interface that allows users to view real-time data, run reports, and receive alerts on temperature deviations.	UI Testing	White Box	Evaluate the web interface's functionality, user-friendliness, and data display accuracy.	Web interface is intuitive and displays data accurately. Users can see Historical and future data.
FR6	The system shall include a framework for testing and validating machine learning models, ensuring their accuracy and	Validation Testing	White Box	Conduct code-level testing for model accuracy and reliability against test datasets.	Models validated, with over 95% accuracy on test datasets

	reliability against test datasets.				
FR7	An automated alert system that notifies users of any detected anomalies or significant temperature deviations.	Functional Testing	Black Box	Check the functionality and reliability of the automated alert system.	Alert system functional, all test deviations notified promptly within 1 minute of deviation detection.

5.4.2. Non-Functional Requirements

Table 9 - Non-Functional Requirements Testing Strategy

Test ID	Description	Testing Strategy	Testing Category	Details of Testing Strategy	Actual Result
NFR1	The system must deliver temperature readings with a minimum accuracy of 99% and ensure system downtime does not exceed 30 minutes per week, guaranteeing reliable and consistent data collection critical for effective analysis.	Performance Testing	White Box	Test system accuracy and ensure downtime does not exceed 30 minutes per week.	System accuracy at 99%, downtime at 0 minutes per week.
NFR2	It should be scalable to accommodate an increasing number of sensors and larger datasets.	Scalability Testing	Black Box	Evaluate the system's ability to handle increased loads and data sizes.	Successfully scaled to handle 2x the initial sensor counts and dataset size.

NFR3	The system should process and analyse data in real-time or near-real-time to enable prompt decision-making.	Performance Testing	Black Box	Assess the system's capability for real-time data processing and analysis.	Data processing and analysis achieved in under 2 seconds.
NFR6	Compliance with data privacy laws and regulations, ensuring that user data is handled confidentially.	Compliance Testing	Black Box	Ensure the system adheres to data privacy standards and regulations.	System fully compliant with GDPR and HIPAA regulations, with no breaches in data handling protocols.
NFR7	The microcontroller and sensors should be optimized for low power consumption, considering the energy constraints in healthcare settings.	Efficiency Testing	White Box	Test the energy efficiency of the microcontroller and sensors in healthcare settings.	The ESP32 and BLE sensors demonstrate exceptionally low power consumption, significantly reducing operational costs without sacrificing performance or connectivity.

5.4.3. LSTM Model Testing Accuracy

In this section, we delve into the testing accuracy of three distinct LSTM models tailored for temperature prediction. These models differ in their architectural configurations, particularly in terms of sequence length and prediction horizon. The findings discussed below shed light on their respective performances, highlighting their capabilities and limitations when faced with unseen data. This analysis is crucial as it underscores the practical utility of the LSTM models in real-world scenarios where accurate and reliable forecasts are essential.

5.4.3.1 Model 1 Analysis

Model 1, configured with a sequence length of 144 and predicting a single future step, exhibits exemplary performance. The RMSE and MAE values stand at 0.15 and 0.08, respectively, demonstrating a high degree of prediction accuracy. Furthermore, this model achieves an impressive 97.72% of predictions within the ±0.5-degree tolerance range. These metrics indicate that Model 1 excels in short-term forecasts, making it highly reliable for applications requiring precise, immediate future predictions. The ability to accurately forecast in the short term can be particularly beneficial in environments where rapid response to temperature changes is critical, such as in controlled climate conditions in pharmaceutical storage or food processing industries.

5.4.3.2 Model 2 Analysis

Model 2, with an increased sequence length of 288 and extending its prediction capability to 12 future steps, shows a slight decrease in performance compared to Model 1, reflected in an RMSE of 0.19 and an MAE of 0.11. Despite this, it still maintains a high accuracy rate, with 96.24% of predictions falling within the ±0.5-degree tolerance. The extended prediction horizon of Model 2 offers valuable foresight, allowing for effective planning and pre-emptive measures over a longer duration. This model's capacity to maintain high accuracy over multiple prediction steps makes it suitable for scenarios where understanding future trends over the next few hours is crucial, such as anticipating temperature fluctuations overnight in greenhouse settings.

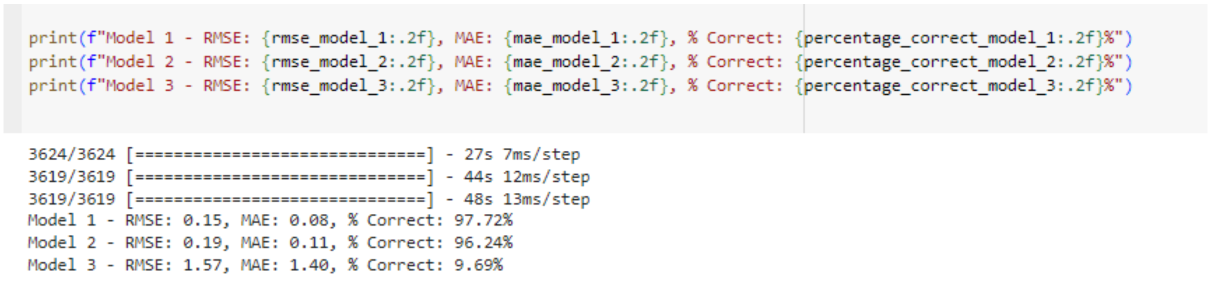
5.4.3.3 Model 3 Analysis

Model 3 also uses a sequence length of 288 but underperforms significantly compared to the other two models, with an RMSE of 1.57 and an MAE of 1.40, indicating less precise predictions. Moreover, only 9.69% of its predictions stay within the established tolerance, showcasing considerable prediction errors. This drop in accuracy and reliability suggests potential issues in model training or the complexity of handling long sequence predictions, which might have led to overfitting or insufficient generalization to unseen data. Such results highlight the challenges and complexities involved in developing LSTM models that maintain accuracy over extended prediction steps, especially when dealing with complex datasets or diverse environmental conditions.

5.4.3.4 Selection and Practical Application

The analysis led to the selection of Model 2 for practical deployment, primarily due to its balance between accuracy and extended prediction capability. Despite its slightly lower accuracy than Model 1, the ability to forecast up to 12 steps ahead provides a more comprehensive outlook, crucial for proactive temperature management. This capability is particularly advantageous in monitoring systems where early warnings can prevent spoilage or ensure that environmental conditions remain within specified limits over time.

The detailed results and visual representation of the models' performance are presented in Figure 21 below, which provides a comparative analysis of the prediction accuracies across the three models. For a deeper exploration of the code and methods used in these evaluations, refer to the full notebook available in Appendix A12.



```
print(f"Model 1 - RMSE: {rmse_model_1:.2f}, MAE: {mae_model_1:.2f}, % Correct: {percentage_correct_model_1:.2f}%")
print(f"Model 2 - RMSE: {rmse_model_2:.2f}, MAE: {mae_model_2:.2f}, % Correct: {percentage_correct_model_2:.2f}%")
print(f"Model 3 - RMSE: {rmse_model_3:.2f}, MAE: {mae_model_3:.2f}, % Correct: {percentage_correct_model_3:.2f}%")
```

```
3624/3624 [==============================] - 27s 7ms/step
3619/3619 [==============================] - 44s 12ms/step
3619/3619 [==============================] - 48s 13ms/step
Model 1 - RMSE: 0.15, MAE: 0.08, % Correct: 97.72%
Model 2 - RMSE: 0.19, MAE: 0.11, % Correct: 96.24%
Model 3 - RMSE: 1.57, MAE: 1.40, % Correct: 9.69%
```

Figure 22 – Model Evaluation & Accuracy Metrics

5.5. Testing Conclusion

This chapter establishes a comprehensive testing framework for the Functional, Non-Functional Requirements and ML of the system. By employing a combination of Black-Box and White-Box testing strategies, the research ensures thorough validation of the system's functionalities and performance. This approach is pivotal in guaranteeing that the system is operationally effective, robust, user-friendly, and compliant with relevant standards.

6. Conclusion

This thesis has successfully explored the integration of Machine Learning techniques within a microcontroller-driven sensor system for enhancing the monitoring of cold chain environments in the pharmaceutical industry. Throughout the course of this study, a comprehensive approach encompassing a literature review, system design and methodology, implementation, and rigorous testing strategy was adopted to ensure a robust analysis and validation of the proposed solution.

Literature Review: The initial phase involved a detailed literature review which provided a foundational understanding of the current technologies and methodologies employed in cold chain monitoring. This review not only highlighted the gaps in current practices but also established the potential for ML applications in enhancing monitoring accuracy and reliability.

Design and Methodology: The design phase focused on developing a scalable and efficient system architecture that could integrate seamlessly with existing technologies. The methodology was carefully crafted to ensure that each component of the system—from data collection through environmental sensors to data transmission via a microcontroller—was optimized for real-time processing and analysis.

Implementation: The implementation phase involved the actual construction of the system, focusing on integrating an ESP32 controller setup that communicated with a Ruuvii sensor. This setup was engineered to transmit sensor data directly to the InfluxDB cloud, providing a robust platform for data storage and management. A Flask web application was developed to interface with InfluxDB, facilitating data representation and enabling the execution of LSTM predictions. This phase was crucial not only for bringing the theoretical designs to life but also for testing the feasibility of the proposed solution in real-world scenarios.

Testing Strategy: A thorough testing strategy was implemented to evaluate both the functional and non-functional requirements of the system. The strategy employed a mix of black-box and white-box testing techniques to ensure comprehensive coverage of all aspects of the system. The functionality of the LSTM models was particularly scrutinized to verify their accuracy in predicting temperature deviations.

6.1. Final Thoughts

The LSTM models demonstrated a high degree of accuracy, with the best-performing model achieving over 96% accuracy in predictions. However, the training and testing of three different models proved to be both time-consuming and resource-intensive, taking over 120 hours and involving significant computational costs. This aspect of the research highlighted the challenges of ML applications in terms of resource and cost efficiency, particularly in settings where multiple models need to be evaluated to determine the most effective solution.

Despite these challenges, the potential for enhancing the system with additional features, such as anomaly detection, remains vast. Future research could explore the integration of anomaly detection models to provide a more comprehensive monitoring tool that not only predicts but also automatically detects and alerts anomalies in real-time. This would further enhance the system's utility, making it an invaluable tool in the pharmaceutical industry's efforts to maintain the integrity of its products throughout the cold chain.

In conclusion, this thesis has laid a solid foundation for the future exploration and implementation of ML techniques in cold chain monitoring systems. It has demonstrated the feasibility and effectiveness of using LSTM models within a microcontroller-driven system, setting the stage for future enhancements that could potentially revolutionize the field.

References

EFPIA. (2022). The Pharmaceutical Industry in Figures. Available: https://www.efpia.eu/media/637143/the-pharmaceutical-industry-in-figures-2022.pdf. [Accessed 23 Oct 2023].

Health Products Regulatory Authority. (2020). ‘Control and Monitoring of Storage and Transportation Temperature Conditions for Medicinal Products and Active Substances’. Available: https://www.hpra.ie/docs/default-source/publications-forms/guidance-documents/ia-g0011-guide-to-control-and-monitoring-of-storage-and-transportation-conditions-v2.pdf. [Accessed 23 Oct 2023].

Gebremariam, E.T., Gebregeorgise, D.T., and Fenta, T.G. (2019) ‘Factors contributing to medicines wastage in public health facilities of South West Shoa Zone, Oromia Regional State, Ethiopia: a qualitative study’, Journal of Pharmaceutical Policy and Practice, 12(1), 29, available: https://doi.org/10.1186/s40545-019-0192-z.

Health Service Executive. (2020). ‘HSE Guidelines for maintenance of cold-chain in vaccine fridges and management of vaccine stock’. Available: https://www.hse.ie/eng/health/immunisation/hcpinfo/vaccineordering/sopnio01.pdf. [Accessed 23 Oct 2023].

Oracle. (2023). What Is Machine Learning? Available: https://www.oracle.com/ie/artificial-intelligence/machine-learning/what-is-machine-learning/. [Accessed 7 Nov 2023].

Microsoft. (2023). Artificial Intelligence vs. Machine Learning. Available: https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/artificial-intelligence-vs-machine-learning. [Accessed 8 Nov 2023].

IBM. (2023). What Is Supervised Learning? Available: https://www.ibm.com/topics/supervised-learning. [Accessed 8 Nov 2023].

Google Cloud. (2023). What Is Unsupervised Learning? Available: https://cloud.google.com/discover/what-is-unsupervised-learning. [Accessed 9 Nov 2023].

Data Camp. (2023). Clustering in Machine Learning: 5 Essential Clustering Algorithms. Available: https://www.datacamp.com/blog/clustering-in-machine-learning-5-essential-clustering-algorithms. [Accessed 9 Nov 2023].

Data Camp. (2023). Introduction to Unsupervised Learning: Types, Applications and Differences from Supervised Learning. Available: https://www.datacamp.com/blog/introduction-to-unsupervised-learning. [Accessed 9 Nov 2023].

Li, Y. (2018). 'Deep Reinforcement Learning: An Overview'. Available: http://arxiv.org/abs/1701.07274. [Accessed 9 Nov 2023].

Data Tonic. (2023). Reinforcement Learning: Bringing Use Cases to Life. Available: https://datatonic.com/insights/reinforcement-learning-identifying-opportunities-use-cases. [Accessed 9 Nov 2023].

MathWorks. (2023). What Is Reinforcement Learning? Available: https://www.mathworks.com/discovery/reinforcement-learning.html. [Accessed 9 Nov 2023].

Feyisa, D., Ejeta, F., Aferu, T., & Kebede, O. (2022). 'Adherence to WHO vaccine storage codes and vaccine cold chain management practices at primary healthcare facilities in Dalocha District of Silt'e Zone, Ethiopia'. Tropical Diseases, Travel Medicine, and Vaccines, 8(1), 10. Available: https://doi.org/10.1186/s40794-022-00167-5.

Airfinity. (2023). Global Wastage of COVID-19 Vaccines Could Be 1.1 Billion Doses. Available: https://airfinity.com/articles/global-wastage-of-covid-19-vaccines-could-be-1-1-billion-doses. [Accessed 10 Nov 2023].

World Health Organization. (n.d.). Cold Chain and logistics management. Available: https://www.who.int/docs/default-source/searo/india/publications/immunization-handbook-107-198-part2.pdf.

Pelican Biothermal. (2023). 2019 Biopharma Cold Chain Logistics Survey Report. Available: https://info.pelicanbiothermal.com/2019tcpsurveyreportpeli-ty-0. [Accessed 10 Nov 2023].

U.S FDA. (2023). 21 CFR 205.50 -- Minimum Requirements for the Storage and Handling of Prescription Drugs and for the Establishment and Maintenance of Prescription Drug Distribution Records. Available: https://www.ecfr.gov/current/title-21/part-205/section-205.50. [Accessed 11 Nov 2023].

U.S FDA. (2023). 21 CFR 203.32 -- Drug Sample Storage and Handling Requirements. Available: https://www.ecfr.gov/current/title-21/part-203/section-203.32. [Accessed 12 Nov 2023].

EUR - LEX. (2023). EUR-Lex - 32002R0178 - EN - EUR-Lex. Available: https://eur-lex.europa.eu/eli/reg/2002/178/oj. [Accessed 12 Nov 2023].

Branch, L.S. (2023). Consolidated Federal Laws of Canada, Food and Drug Regulations. Available: https://laws-lois.justice.gc.ca/eng/Regulations/C.R.C.,_c._870/index.html. [Accessed 12 Nov 2023].

Caroline, B. (2020) Q&A: Cold Chains, COVID-19 Vaccines and Reaching Low-Income Countries | Imperial News | Imperial College London [online], Imperial News, available: https://www.imperial.ac.uk/news/209993/qa-cold-chains-covid-19-vaccines-reaching/ [accessed 12 Nov 2023].

IBM (2023) IoT for Blockchain Cold Chain - IBM Cloud Architecture Center [online], available: https://www.ibm.com/cloud/architecture/architectures/iot-blockchain-cold-chain/ [accessed 13 Nov 2023].

GlobalSpec (2023) Automation and Precision: The Role of Smart Sensors | GlobalSpec [online], available:

https://insights.globalspec.com/article/21412/automation-and-precision-the-role-of-smart-sensors [accessed 13 Nov 2023].

What Is an Embedded System? [online] (2023) IoT Agenda, available: https://www.techtarget.com/iotagenda/definition/embedded-system [accessed 15 Nov 2023].

Microcontroller vs Microprocessor - What Are the Differences? [online] (2019) Total Phase Blog, available: https://www.totalphase.com/blog/2019/12/microcontroller-vs-microprocessor-what-are-the-differences/ [accessed 14 Nov 2023].

Hackaday (2020) 'The TMS1000: The First Commercially Available Microcontroller', Hackaday, available: https://hackaday.com/2020/02/18/the-tms1000-the-first-commercially-available-microcontroller/ [accessed 14 Nov 2023].

TMS 1000 4-Bit Microcontroller, TI, 1976 - CHM Revolution [online] (2023) available: https://www.computerhistory.org/revolution/digital-logic/12/284/1564 [accessed 14 Nov 2023].

Intel (2023) The Story of the Intel® 4004 [online], Intel, available: https://www.intel.com/content/www/us/en/history/museum-story-of-intel-4004.html [accessed 14 Nov 2023].

Creative, V. and Bluefruit (2022) A Brief History of Embedded Operating Systems [online], Bluefruit Software, available: https://bluefruit.co.uk/processes/a-brief-history-of-embedded-operating-systems/ [accessed 15 Nov 2023].

Kogatam, S. (n.d.) 'The Birth, Evolution and Future of Microprocessor', available: https://www.sjsu.edu/people/robert.chun/courses/CS247/s4/M.pdf.

Car Rental Gateway (2020) What Is an Electronic Control Unit (ECU) in a Car? [online], Car Rental Glossary, available: https://www.carrentalgateway.com/glossary/electronic-control-unit/ [accessed 15 Nov 2023].

Promwad (2023) Wireless Technologies in Embedded Systems [online], Promwad, available: https://promwad.com/technologies/wireless-technologies [accessed 15 Nov 2023].

Evans, H. (2023) How IoT Embedded Systems Are Impacting Industries [online], Velvetech, available: https://www.velvetech.com/blog/iot-embedded-systems/ [accessed 15 Nov 2023].

Almusallam, S.S. (2015) 'Power Consumption in the Embedded System', 3(8), available: https://www.ijese.org/wpcontent/uploads/Papers/v3i8/H0997063815.pdf

Walls, C. (2015) 'Power management in embedded software', Embedded.com, available: https://www.embedded.com/power-management-in-embedded-software/ [accessed 15 Nov 2023].

NASA (2023) Apollo Flight Journal - The Apollo On-Board Computers [online], available: https://history.nasa.gov/afj/compessay.html [accessed 15 Nov 2023].

Harvie, L. (2023) 'Digital Signal Processing (DSP) in Embedded Systems', Medium, available: https://medium.com/@lanceharvieruntime/digital-signal-processing-dsp-in-embedded-systems-68b4649ff441 [accessed 16 Nov 2023].

Components 101 (2023) Introduction to Analog to Digital Converters (ADC Converters) [online], Components101, available: https://components101.com/articles/analog-to-digital-adc-converters [accessed 16 Nov 2023].

Crane, L. (2023) Back to Analog Computing: Merging Analog and Digital Computing on a Single Chip | Electrical Engineering [online], available: https://www.ee.columbia.edu/back-analog-computing-merging-analog-and-digital-computing-single-chip [accessed 16 Nov 2023].

Embedded Systems Lab (n.d.) 'ADC devices | Embedded systems', available: https://labs.dese.iisc.ac.in/embeddedlab/adc-devices/ [accessed 16 Nov 2023].

Jasani, M. (2021) Minimizing the Challenges in Embedded Systems Security [online], Excellent Webworld, available: https://www.excellentwebworld.com/embedded-systems-security/ [accessed 16 Nov 2023].

Savjani, R. (2018) ‘6 Critical Challenges Facing the Embedded Systems Security’, available: https://www.einfochips.com/blog/6-critical-challenges-facing-the-embedded-systems-security/ [accessed 16 Nov 2023].

Media, O. (2023) 5 Steps to Secure Embedded Software [online], Embedded Computing Design, available: https://embeddedcomputing.com/technology/security/5-steps-to-secure-embedded-software-2 [accessed 16 Nov 2023].

Chapter 20: Analog to Digital Conversion [Analog Devices Wiki] [online] (2023) available: https://wiki.analog.com/university/courses/electronics/text/chapter-20 [accessed 16 Nov 2023].

Vaisala (2021) Silent, Deadly, but Vital – CO2 and the Pharma Logistics Cold Chain | Vaisala [online], available: https://www.vaisala.com/en/blog/2021-02/silent-deadly-vital-co2-and-pharma-logistics-cold-chain [accessed 16 Nov 2023].

Sensitech (2023) The Ultimate Guide to Cold Chain Temperature Monitoring | Sensitech Blog [online], Sensitech, available: https://www.sensitech.com/en/blog/blog-articles/blog-ultimate-guide-cold-chain-monitoring.html [accessed 16 Nov 2023].

LabRepCo (n.d.) ‘Pfizer / BioNTech COVID-19 Vaccine Storage Capacity for Ultra-Low Temperature Freezers (-70°C/-94°F)’, LabRepCo, LLC, available: https://www.labrepco.com/pfizer-biontech-covid-19-vaccine-storage-capacity-for-ultra-low-temperature-freezers-86c/ [accessed 17 Nov 2023].

Hygroscopic Pharmaceuticals [online] (2023) available: https://www.neutroplast.com/news/neutroplast/en/146 [accessed 17 Nov 2023].

medsage.gov.nz (2023) Medicine Storage - An Uncontained Issue? [online], available: https://www.medsafe.govt.nz/profs/PUArticles/September2014MedicineStorage.htm [accessed 17 Nov 2023].

Habas, C. (2023) How Does a Door Sensor Work? [online], SafeWise, available: https://www.safewise.com/home-security-faq/how-door-sensors-work/ [accessed 17 Nov 2023].

Door Open Sensors: Applications for Fleets [online] (2023) available: https://www.samsara.com/guides/door-open-sensors/ [accessed 17 Nov 2023].

#01 What's a Magnetic Sensor? [online] (2023) Tutorials | Hall Sensors | Products | Asahi Kasei Microdevices (AKM), available: https://www.akm.com/content/www/akm/eu/en/products/hall-sensor/tutorial/magnetic-sensor.html [accessed 17 Nov 2023].

Processing, C.T. (2020) 'Cryogenic Temperature: Everything You Need to Know - Controlled Thermal Processing', Controlled Thermal Processing - Reduce costs by increasing Performance, available: https://ctpcryogenics.com/cryogenic-temperature-everything-you-need-to-know/ [accessed 17 Nov 2023].

TechTarget (2023) What Are Sensors and How Do They Work? [online], WhatIs.com, available: https://www.techtarget.com/whatis/definition/sensor [accessed 17 Nov 2023].

McGrath, M.J. and Scanaill, C.N. (2013) 'Regulations and Standards: Considerations for Sensor Technologies', in McGrath, M.J. and Scanaill, C.N., eds., Sensor Technologies: Healthcare, Wellness, and Environmental Applications, Berkeley, CA: Apress, 115–135, available: https://doi.org/10.1007/978-1-4302-6014-1_6.

Gemini Data Loggers (2023) Temperature Monitoring in Pharmaceutical Warehouses | Tinytag Loggers [online], geminidataloggers.com, available:

https://www.geminidataloggers.com/info/temperature-rh-monitoring-pharmaceutical-warehouse [accessed 17 Nov 2023].

mikeej (2021) Calibration within the Pharmaceutical Industry - What You Need to Know - Calibration Select by Avery Weigh-Tronix [online], Calibration Select, available: https://calibrationselect.co.uk/blog/blog/calibration-within-the-pharmaceutical-industry-what-you-need-to-know/ [accessed 17 Nov 2023].

Inc, T.C. (2022) 'Why Are Pharmacies Deploying IoT Cold Chain Monitoring Solutions?', IoT For All, available: https://www.iotforall.com/why-are-pharmacies-deploying-iot-cold-chain-monitoring-solutions [accessed 17 Nov 2023].

Velos (2023) Why Is Cellular IoT-Based Pharmaceutical Cold Chain Management Important? [online], available: https://blog.velosiot.com/iot-in-pharmaceutical-cold-chain [accessed 17 Nov 2023].

tableau (2023) Time Series Analysis: Definition, Types & Techniques | Tableau [online], available: https://www.tableau.com/learn/articles/time-series-analysis [accessed 21 Nov 2023].

Gupta, S. (2022) What Is Time Series Forecasting? Overview, Models & Methods [online], Springboard Blog, available: https://www.springboard.com/blog/data-science/time-series-forecasting/ [accessed 21 Nov 2023].

Pandian, S. (2021) 'Time Series Analysis and Forecasting | Data-Driven Insights (Updated 2023)', Analytics Vidhya, available: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-to-time-series-analysis/ [accessed 21 Nov 2023].

National Institute of Standards and Technology (2023) 6.4.1. Definitions, Applications and Techniques [online], available: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc41.htm [accessed 21 Nov 2023].

Nathaniel, J. (2021) Introduction to ARIMA for Time Series Forecasting [online], Medium, available: https://towardsdatascience.com/introduction-to-arima-for-time-series-forecasting-ee0bc285807a [accessed 21 Nov 2023].

Chao, D.-Y. (2021) ‘Performing Time Series Analysis using ARIMA Model in R’, Analytics Vidhya, available: https://www.analyticsvidhya.com/blog/2021/11/performing-time-series-analysis-using-arima-model-in-r/ [accessed 21 Nov 2023].

Analytics Vidhya (2023) Introduction to ARIMA Models [online], available: https://www.analyticsvidhya.com/blog/2021/11/performing-time-series-analysis-using-arima-model-in-r/ [accessed 21 Nov 2023].

Nathaniel, J. (2021) Introduction to ARIMA for Time Series Forecasting [online], Medium, available: https://towardsdatascience.com/introduction-to-arima-for-time-series-forecasting-ee0bc285807a [accessed 21 Nov 2023].

Biswal, A. (2023) Power of Recurrent Neural Networks (RNN): Revolutionizing AI [online], Simplilearn.com, available: https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn [accessed 22 Nov 2023].

Yasar, K. (2023) What Is a Neural Network? Definition, Types and How It Works [online], Enterprise AI, available: https://www.techtarget.com/searchenterpriseai/definition/neural-network [accessed 22 Nov 2023].

Kalita, D. (2022) ‘A Brief Overview of Recurrent Neural Networks (RNN)’, Analytics Vidhya, available: https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/ [accessed 22 Nov 2023].

Wang, P. (2023) Figure 5: The Inner Structure of LSTM Cell. Γ f , Γu, Γo Are Sigmoid... [online], ResearchGate, available: https://www.researchgate.net/figure/The-inner-structure-of-LSTM-cell-G-f-Gu-Go-are-sigmoid-functions-of-the-forget-gate_fig4_344779271 [accessed 22 Nov 2023].

Whitfield, B. (2023) What Are Recurrent Neural Networks? | Built In [online], available: https://builtin.com/data-science/recurrent-neural-networks-and-lstm [accessed 23 Nov 2023].

Saxena, S. (2021) 'What is LSTM? Introduction to Long Short-Term Memory', Analytics Vidhya, available: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/ [accessed 23 Nov 2023].

Brownlee, J. (2017) 'A Gentle Introduction to Backpropagation Through Time', MachineLearningMastery.com, available: https://machinelearningmastery.com/gentle-introduction-backpropagation-time/ [accessed 23 Nov 2023].

Smith, G.M. (2023) What Is Signal Processing? [online], Data Acquisition | Test and Measurement Solutions, available: https://dewesoft.com/blog/what-is-signal-processing [accessed 23 Nov 2023].

GlobalSpec (2023) Temperature Probes Selection Guide: Types, Features, Applications | GlobalSpec [online], available: https://www.globalspec.com/learnmore/sensors_transducers_detectors/temperature_sensing/temperature_probes [accessed 14 Dec 2023].

McKee, A. (2023) A Data Scientist's Guide to Signal Processing [online], available: https://www.datacamp.com/tutorial/a-data-scientists-guide-to-signal-processing [accessed 23 Nov 2023].

Automata (2023) 'Focus on Signal Processing in Data Science', **AI monks.io**, available: https://medium.com/aimonks/focus-on-signal-processing-in-data-science-198b24992643 [accessed 23 Nov 2023].

Lawton, G. (2023) Data Preprocessing: Definition, Key Steps and Concepts [online], Data Management, available:

https://www.techtarget.com/searchdatamanagement/definition/data-preprocessing [accessed 23 Nov 2023].

Garberman, B. (2020) How to Clean Up Noisy Sensor Data With a Moving Average Filter | Arduino [online], Maker Pro, available: https://maker.pro/arduino/tutorial/how-to-clean-up-noisy-sensor-data-with-a-moving-average-filter [accessed 23 Nov 2023].

Skan Editorial Staff (2023) How to Use Machine Learning to Separate the Signal from the Noise | Skan [online], available: https://www.skan.ai/process-mining-insights/how-to-use-machine-learning-to-separate-the-signal-from-the-noise [accessed 23 Nov 2023].

ESP32 Wi-Fi & Bluetooth SoC | Espressif Systems [online] (2023) available: https://www.espressif.com/en/products/socs/esp32 [accessed 24 Nov 2023].

Last Minute Engineers (2018) Insight Into ESP32 Sleep Modes & Their Power Consumption [online], Last Minute Engineers, available: https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/ [accessed 24 Nov 2023].

Lextrait, T. (2016) Arduino: Power Consumption Compared • Thomas Lextrait [online], Thomas Lextrait on Svbtle, available: https://tlextrait.svbtle.com/arduino-power-consumption-compared [accessed 24 Nov 2023].

Raspberry Pi Ltd (2023) Buy a Raspberry Pi Pico [online], Raspberry Pi, available: https://www.raspberrypi.com/products/raspberry-pi-pico/ [accessed 24 Nov 2023].

‘Raspberry Pi Pico Datasheet’ (n.d.) available: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf

UNO R3 | Arduino Documentation [online] (2023) available: https://docs.arduino.cc/hardware/uno-rev3 [accessed 24 Nov 2023].

Das, S. (2023) What Is Black Box Testing: Types, Tools & Examples [online], BrowserStack, available: https://browserstack.wpengine.com/guide/black-box-testing/ [accessed 6 Dec 2023].

Hamilton, T. (2023) White Box Testing – What Is, Techniques, Example & Types [online], available: https://www.guru99.com/white-box-testing.html [accessed 6 Dec 2023].

Appendices

Appendix A: Code Listing

```
# Import Google Drive For JSON File
from google.colab import drive
drive.mount('/content/drive')



# Import JSON file for Pre-processing
import json

file_path = '/content/drive/MyDrive/PreProcessedData2.json'
with open(file_path, 'r') as file:
    data = json.load(file)
```

A1 – Mounting Google Drive & Importing JSON File

```
import numpy as np
# Define the number of standard deviations for the outlier threshold
n_std_devs = 2

# Function to replace outliers with NaN, based on each sensor's data
def replace_outliers(data, n_std_devs=2):
    # Calculate mean and standard deviation for the sensor's data
    mean = data.mean()
    std = data.std()

    # Define cutoff values for outliers
    cutoff = n_std_devs * std
    lower, upper = mean - cutoff, mean + cutoff

    # Replace values outside the cutoff range with NaN
    data = np.where((data < lower) | (data > upper), np.nan, data)
    return data

# Apply the outlier replacement function to each sensor's data
for sensor in df['channel_description'].unique():
    # Identify the data for the current sensor
    sensor_indices = df['channel_description'] == sensor

# Replace outliers in channel_value column for this sensor with NaN
    df.loc[sensor_indices, 'channel_value'] =
replace_outliers(df.loc[sensor_indices, 'channel_value'], n_std_devs)

# Index Date column
if 'date' in df.columns:
    df['date'] = pd.to_datetime(df['date'])
    df.set_index('date', inplace=True)

# Perform interpolation
df['channel_value'] =
df.groupby('channel_description')['channel_value'].transform(lambda
x: x.interpolate(method='time'))
```

A2 - Outlier Removal Based on Standard Deviation & Interpolation to Remove Nan Values.

```python
# Determine the predominant category for each sensor
threshold = 0.5  # 50% threshold
predominant_categories_filtered = {}


for sensor, proportions in category_proportions.items():
    if not proportions:  # If no readings fall within any category
        sensors_dropped_count += 1
        continue  # Skip this sensor
    max_category = max(proportions, key=proportions.get)
    if proportions[max_category] >= threshold:
        predominant_categories_filtered[sensor] = max_category
        sensors_remaining_count += 1
    else:
        predominant_categories_filtered[sensor] = 'other'
        sensors_dropped_count += 1


# Apply the predominant category to each sensor's records
df['predominant_category'] = df['channel_description'].apply(lambda x:
predominant_categories_filtered.get(x, 'other'))
```

A3 – Determining Predominant Operational Categories for Sensors Based on Temperature Range Analysis

```
# Ensure df is sorted by 'date' to maintain chronological order
df.sort_values(by='date', inplace=True)

final_df = pd.DataFrame()

# Calculate the total duration needed for each sensor
resample_interval = pd.Timedelta('5T')  # 5 minutes
max_timestamp = pd.Timestamp.min

for category in df['predominant_category'].unique():
    category_df = df[df['predominant_category'] == category]
    sensor_means = category_df.groupby('channel_description')['channel_value'].mean()
    sorted_sensors = sensor_means.sort_values().index

#Nested For Loop
for sensor in sorted_sensors:
        sensor_df = category_df[category_df['channel_description'] == sensor].copy()

        if sensor_df.empty:
            continue

        # Calculate number of points and total duration for this sensor
        num_points = len(sensor_df)
        total_duration = (num_points - 1) * resample_interval

        # Create new timestamps starting from max_timestamp +resample_interval
        new_timestamps = pd.date_range(start=max_timestamp +    resample_interval,
periods=num_points, freq=resample_interval)

        # Update max_timestamp for the next sensor
        max_timestamp = new_timestamps[-1]

        # Update sensor_df with new timestamps
        sensor_df.index = new_timestamps

        # Process and append this sensor's data to final_df
        temp_categorical = sensor_df.iloc[0][['channel_unit', 'channel_description',
'category', 'predominant_category']]
        sensor_df = sensor_df.reset_index().rename(columns={'index': 'date'})
        sensor_df[['channel_unit', 'channel_description', 'category',
'predominant_category']] = temp_categorical
        final_df = pd.concat([final_df, sensor_df])
```

A4 – Aggregating & Setting a Consistent 5 Minute Frequency for the Time Series

```
# Predefined parameter sets to try
parameter_sets = [
    (5,1,6),
    # Add more parameter sets as needed
]

best_aic = np.inf
best_model = None
best_params = None

# Iterate over parameter sets, fitting an ARIMA model for each and tracking
the best AIC
for params in parameter_sets:
    p, d, q = params
    model = ARIMA(channel_value_filled, order=(p, d, q))
    model_fit = model.fit()

    if model_fit.aic < best_aic:
        best_aic = model_fit.aic
        best_model = model_fit
        best_params = params
        print(f"New best model found: ARIMA{params} with AIC: {best_aic}")
    else:
        print(f"Model ARIMA{params} with AIC: {model_fit.aic} did not
improve.")

# Print the summary
print(best_model.summary())
```

A5 - ARIMA Model with Pre-Defined Parameters

```
# Imports LSTM Model Training
import os
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.models import load_model
from keras_tuner import RandomSearch
```

A6 – Imports for LSTM Model Training

```
# Function to build the LSTM model architecture with hyperparameter tuning options
def build_model(hp):
    model = keras.Sequential()
    # Option to add more LSTM layers
    model.add(layers.LSTM(
        units=hp.Int('units', min_value=32, max_value=512, step=32),
        return_sequences=True,
        input_shape=(sequence_length, 1))
    )
    # Adding the first LSTM layer with hyperparameter options for # of units
    model.add(layers.LSTM(
        units=hp.Int('units', min_value=32, max_value=512, step=32),
        return_sequences=False)
    )
    # Dense layer with hyperparameter options for the number of units
    model.add(layers.Dense(
        units=hp.Int('dense_units', min_value=16, max_value=128, step=16),
        activation='relu')
    )
    # Output layer
    model.add(layers.Dense(1))
    # Model compilation with an Adam optimizer and learning rate tuning
    model.compile(
        optimizer=keras.optimizers.Adam(
            hp.Float('learning_rate', min_value=1e-4, max_value=1e-2, sampling='LOG')
        ),
        loss='mean_squared_error'
    )
    return model
```

A7 - Function to Build the LSTM Model Architecture with Hyperparameter Tuning Options

```
# Loop through each category of data
for category in categories:
    print(f"Processing category: {category}")
    # Filter dataframe for the current category
    category_df = df[df['category'] == category]

    # Prepare sequences and labels for LSTM model
    sequences = []
    labels = []
    # Generate sequences of 144 time steps (equivalent to 12 hours of data at 5-minute
intervals)
    for i in range(sequence_length, len(category_df)):
        sequences.append(category_df['channel_value_normalized'].values[i -
sequence_length:i])
        labels.append(category_df['channel_value_normalized'].values[i])

    # Convert sequences and labels into numpy arrays for model training
    sequences = np.array(sequences)
    labels = np.array(labels)
    # Reshape sequences for LSTM input (samples, time steps, features)
    sequences = np.reshape(sequences, (sequences.shape[0], sequences.shape[1], 1))

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(sequences, labels, test_size=0.2,
random_state=42)

    # Define directory for saving model checkpoints during training
    checkpoint_dir = f'/content/drive/My Drive/keras_tuner_checkpoints/{category}/'
    os.makedirs(checkpoint_dir, exist_ok=True)

    # Setup model checkpoint callback to save only the best model based on validation loss
    checkpoint_filepath = checkpoint_dir + 'model_best.h5'
    checkpoint_callback = ModelCheckpoint(
        filepath=checkpoint_filepath,
        save_weights_only=True,
        monitor='val_loss',
        mode='min',
        save_best_only=True
    )

    # Initialize Keras Tuner for hyperparameter optimization using Random Search
    tuner = RandomSearch(
        build_model,
        objective='val_loss',
        max_trials=5,  # Limit the search to 5 trials
        executions_per_trial=1,
        directory=f'/content/drive/My Drive/keras_tuner_{category}',
        project_name=f'lstm_tuning_{category}',
        overwrite=False
    )

    # Start the hyperparameter search process
    tuner.search(
        X_train, y_train,
        epochs=10,
        validation_data=(X_test, y_test),
        callbacks=[
            EarlyStopping(monitor='val_loss', patience=3),  # Stop early if no improvement
            checkpoint_callback  # Save the best model during the search
        ]
    )

    # Retrieve the best hyperparameters and rebuild the model
    best_hp = tuner.get_best_hyperparameters(num_trials=1)[0]
    best_model = build_model(best_hp)
    best_model.load_weights(checkpoint_filepath)  # Load the best model weights

    # Evaluate the best model on the test set
    predictions = best_model.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, predictions))
    print(f"Best RMSE for {category}: {rmse}")

    # Save the fully configured model, including its architecture and weights
    best_model.save(f'/content/drive/My
Drive/keras_tuner_checkpoints/{category}/final_model_{category}.h5')

    # Output a summary of the hyperparameter tuning results
    tuner.results_summary()
```

A8 - Applying the LSTM Hyperparameter Tuning Model to Various Categories Within the Dataset

```python
# This extracts all unique sensor identifiers except the last one for training.
train_sensors = df['channel_description'].unique()[:-1]

# The last unique sensor identifier is reserved for testing.
test_sensor = df['channel_description'].unique()[-1]

# Filter the dataframe to only include rows from the sensors selected for training.
train_df = df[df['channel_description'].isin(train_sensors)]

# Filter the dataframe to only include rows from the sensor selected for testing.
test_df = df[df['channel_description'] == test_sensor]

def create_sequences(data, sequence_length, prediction_steps):
    sequences = []  # Initialize a list
    labels = []  # Initialize a list to hold the predicted values.
    # Loop over the dataset to create sequences and corresponding predictions/labels
    for i in range(sequence_length, len(data) - prediction_steps + 1):
        sequences.append(data['channel_value_normalized'][i - sequence_length:i].values)
        labels.append(data['channel_value_normalized'][i:i + prediction_steps].values)

# Convert the lists of sequences and labels into Numpy arrays for model
training/testing.
    return np.array(sequences), np.array(labels)

sequence_length = 144  # how many past values will be used to predict future values - 12
hours
prediction_steps = 3   # Defines how many future values will be predicted.

# Generate training sequences and labels using the training dataset.
X_train, y_train = create_sequences(train_df, sequence_length, prediction_steps)

# Generate testing sequences and labels using the testing dataset.
X_test, y_test = create_sequences(test_df, sequence_length, prediction_steps)

# Reshape the input sequences to be 3-dimensional, as required by the LSTM model.
# The shape is [samples, timesteps, features], with 'features' set to 1.
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

A9 - Split the Data (Training and Test) and Create Sequences for the LSTM Model

```
def build_model():
    # Initialize a Sequential model
    model = keras.Sequential([
 # Add a first LSTM layer with 224 units, specifying input shape based on previous
hyperparameter tuning
        layers.LSTM(
         units=224,# Number of units/neurons in this layer, determined from trial
         return_sequences=True,# Whether to return the full sequence to the next layer
         input_shape=(144, 1)),# Shape of input data, 144 time steps with 1 feature each

   # Add a second LSTM layer with 224 units, not returning the full sequence this time
        layers.LSTM(
          units=224,  # Keeping the same number of units
          return_sequences=False),

# Add a Dense layer with 80 units and ReLU activation function, as determined from trial
        layers.Dense(
            units=80,  # Number of units/neurons in this layer
            activation='relu'),
        # Add a final Dense output layer with 1 unit for predicting a single value
        layers.Dense(1)  # Output layer for prediction
    ])
    # Compile the model with Adam optimizer and mean squared error loss function
    model.compile(
     optimizer=keras.optimizers.Adam(learning_rate=0.0016625414224592842),  #specific
learning rate from trial 4
        loss='mean_squared_error'  # Loss function for regression
    )
    return model

# Instantiate the model using the predefined architecture and hyperparameters
model = build_model()

# Train the model using the training data
history = model.fit(
    X_train, y_train,
    epochs=10,  # Number of epochs to train for
    batch_size=32,  # Batch size during training
    validation_data=(X_test, y_test),  # Validation data to monitor model's performance
on unseen data
    verbose=1 # Show training progress for each epoch
)
```

A10 – Initial Model Ready for Training & Validation After Trial Runs to Find Optimal Parameters

```
# Load the pre-trained models
model_1 = keras.models.load_model('/content/drive/MyDrive/LSTMModelSaved/')
model_2 = keras.models.load_model('/content/drive/MyDrive/LSTMModelSaved2/')
model_3 = keras.models.load_model('/content/drive/MyDrive/LSTMModelSaved3/final_model')

# The sequence lengths as used during the training of all models
sequence_length_model_1 = 144
sequence_length_model_2 = 288
sequence_length_model_3 = 288

# Unnormalize values for readable metrics
def unnormalize(values, min_val, max_val):
    """Un-normalize an array of values."""
    return values * (max_val - min_val) + min_val


# Prepare the data for the LSTM models
def prepare_sequences(dataframe, sequence_length, prediction_steps=1):
    sequences = []
    target = []
    for i in range(len(dataframe) - sequence_length - prediction_steps + 1):
        sequences.append(dataframe['channel_value_normalized'].iloc[i:i +
sequence_length].values)
        target.append(dataframe['channel_value_normalized'].iloc[i + sequence_length:i +
sequence_length + prediction_steps].values)
    return np.array(sequences), np.array(target)

# Extract the sequences from the dataframe for all models
X_test_model_1, y_test_model_1 = prepare_sequences(test_df, sequence_length_model_1)
X_test_model_2, y_test_model_2 = prepare_sequences(test_df, sequence_length_model_2,
prediction_steps=12)
X_test_model_3, y_test_model_3 = prepare_sequences(test_df, sequence_length_model_3,
prediction_steps=12)

# Reshape the data for LSTM input
X_test_model_1 = np.reshape(X_test_model_1, (X_test_model_1.shape[0],
sequence_length_model_1, 1))
X_test_model_2 = np.reshape(X_test_model_2, (X_test_model_2.shape[0],
sequence_length_model_2, 1))
X_test_model_3 = np.reshape(X_test_model_3, (X_test_model_3.shape[0],
sequence_length_model_3, 1))
```

```python
# All but one sensor for training
train_sensors = df['channel_description'].unique()[:-1]
test_sensor = df['channel_description'].unique()[-1]

train_df = df[df['channel_description'].isin(train_sensors)]
test_df = df[df['channel_description'] == test_sensor]

def create_sequences(data, sequence_length, prediction_steps):
    sequences = []
    labels = []
    for i in range(sequence_length, len(data) - prediction_steps + 1):
        sequences.append(data['channel_value_normalized'][i - sequence_length:i].values)
        labels.append(data['channel_value_normalized'][i:i + prediction_steps].values)
    return np.array(sequences), np.array(labels)

# Adjusted sequence_length to 288
sequence_length = 288  # Now capturing 24 hours of data at 5-minute intervals
prediction_steps = 12  # Forecasting 12 future steps

# new training and testing sequences with the updated sequence_length and
prediction_steps
X_train, y_train = create_sequences(train_df, sequence_length, prediction_steps)
X_test, y_test = create_sequences(test_df, sequence_length, prediction_steps)

# Reshape for LSTM input
X_train = X_train.reshape((X_train.shape[0], sequence_le ngth, 1))
X_test = X_test.reshape((X_test.shape[0], sequence_length, 1))

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow import keras
from tensorflow.keras import layers
import os

# Ensure drive is mounted
drive.mount('/content/drive', force_remount=True)

def build_model(sequence_length, prediction_steps):
    # Adjusted the learning rate in the model compilation
    model = keras.Sequential([
        layers.LSTM(units=224, return_sequences=True, input_shape=(sequence_length, 1)),
        layers.LSTM(units=224),
        layers.Dense(units=80, activation='relu'),
        layers.Dense(prediction_steps)
    ])
    model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0016625414224592842),
                  loss='mean_squared_error')
    return model

sequence_length = 288
prediction_steps = 12
```

```
batch_size = 128

checkpoint_path = '/content/drive/My Drive/model_weights.h5'
if os.path.exists(checkpoint_path):
    print("Checkpoint found. Loading the model...")
    model = keras.models.load_model(checkpoint_path)
else:
    print("Checkpoint not found. Building a new model...")
    model = build_model(sequence_length, prediction_steps,
learning_rate=0.0016625414224592842)

early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
checkpoint = ModelCheckpoint(checkpoint_path, monitor='val_loss', verbose=1,
save_best_only=True, mode='min')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_lr=0.0001,
verbose=1)

#Resume from the last completed epoch
initial_epoch = 33  # Adjust based on the last epoch completed

history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=batch_size,
    validation_data=(X_test, y_test),
    callbacks=[early_stopping, checkpoint, reduce_lr],
    initial_epoch=initial_epoch,
    verbose=1
)
```

A11 – LSTM Model 2

```
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error
from math import sqrt

def evaluate_model(model, X_test, y_test, min_val, max_val, prediction_steps=1,
tolerance=0.5):
    predictions = model.predict(X_test)
    if prediction_steps != 1:
        predictions = predictions[:, -prediction_steps:]

    predictions_unnorm = unnormalize(predictions, min_val, max_val)
    y_test_unnorm = unnormalize(y_test, min_val, max_val)

    assert predictions_unnorm.shape == y_test_unnorm.shape, "The shape of predictions and
actual values must match."

    rmse = sqrt(mean_squared_error(y_test_unnorm, predictions_unnorm))
    mae = mean_absolute_error(y_test_unnorm, predictions_unnorm)

    # Calculate the percentage of correct predictions within the tolerance
    correct_predictions = np.abs(predictions_unnorm - y_test_unnorm) <= tolerance
    percentage_correct = np.mean(correct_predictions) * 100  # Convert to percentage

    return rmse, mae, percentage_correct

# Evaluate each model and get RMSE, MAE, and percentage of correct predictions
rmse_model_1, mae_model_1, percentage_correct_model_1 = evaluate_model(
    model_1, X_test_model_1, y_test_model_1, -85.199997, 37.700000)
rmse_model_2, mae_model_2, percentage_correct_model_2 = evaluate_model(
    model_2, X_test_model_2, y_test_model_2, -85.199997, 37.700000, prediction_steps=12)
rmse_model_3, mae_model_3, percentage_correct_model_3 = evaluate_model(
    model_3, X_test_model_3, y_test_model_3, -85.199997, 37.700000, prediction_steps=12)

print(f"Model 1 - RMSE: {rmse_model_1:.2f}, MAE: {mae_model_1:.2f}, % Correct:
{percentage_correct_model_1:.2f}%")
print(f"Model 2 - RMSE: {rmse_model_2:.2f}, MAE: {mae_model_2:.2f}, % Correct:
{percentage_correct_model_2:.2f}%")
print(f"Model 3 - RMSE: {rmse_model_3:.2f}, MAE: {mae_model_3:.2f}, % Correct:
{percentage_correct_model_3:.2f}%")
```

A12 – Testing of LSTM Models

```
# Initialize InfluxDB client

client = InfluxDBClient(url=INFLUXDB_URL, token=INFLUXDB_TOKEN, org=INFLUXDB_ORG)

query_api = client.query_api()



# Query InfluxDB to retrieve the last 30 days of temperature data

   query = f'from(bucket: "{INFLUXDB_BUCKET}") |> range(start: -30d) |> filter(fn: (r) =>
r._measurement ==
"B0A732FFFFF1A160_BLE_0_0_0_E2C4F3FFFF199E64_RuuviTag_0_Temperature") |> filter(fn: (r)
=> exists r._value)'

   result = query_api.query(org=INFLUXDB_ORG, query=query)




   # Process the results into a DataFrame and sort by timestamp

   data = [

      (record.get_time(), record.get_value())

      for table in result

      for record in table.records

   ]

   df = pd.DataFrame(data, columns=["timestamp", "value"]).sort_values(by="timestamp")
```

A13 – Initialization of influxDB & Query to Fetch Temperature Data