Thermal Comfort Prediction Model for a Bio Sauna

Sonali Liyana Badalage, Alexis Chambers,
Piyumi Weebadu Arachchige, Emilia Pyyny-Polat

19/12/2024

GitHub Repository: https://github.com/thermal-comfort-prediction-model

1 Introduction

For this project, humidity, temperature, and pressure data were collected, processed, and visualized to predict the comfort of a bio sauna. A traditional Finnish sauna has temperatures from 80-100°C and humidity of 10-20% [5]. A bio sauna differs from a traditional Finnish sauna in its lower temperature range of 50-60°C and higher humidity levels of up to 50% [3]. A bio sauna is meant to be used for a longer period of time than a traditional sauna. Automatic adjustment of the environment over time using our sensor model would make for an uninterrupted therapeutic experience. We also chose to create a model for a bio sauna for its lower temperature range which was more compatible with our devices' operating limits [4].

1. **Sensor Data Collection:**
   We collected data from sensors using the Raspberry Pi Pico W. The sensors used in this project include the BMP280 for temperature and barometric pressure and the DHT22 for measuring temperature and humidity. Then the data is labeled into two categories like "Healthy" and "Unhealthy" based on temperature, humidity and pressure values in the bio-sauna. Since temperature values are obtained from both sensors, the mean value will be considered.

2. **Data Processing:**
   We combined the data into a single file and defined labels based on the thresholds shown in the Table 1. To preprocess the data, add noise to simulate sensor variability (e.g., ±2°C, ±5% RH, ±2 hPa) and balance the dataset by ensuring an equal number of "Healthy" and "Unhealthy" samples. Next, we scaled the temperature, humidity, and pressure values to a [0, 1] range to improve model performance and ensure the ML

model treats them as equally important. The StandardScaler was used to normalize the data.

Table 1: Threshold values

Temperature (°C)	**Humidity (%RH)**	**Pressure (hPa)**	**Status**
50 – 65	40 – 55	999 – 1020	Healthy
<50 or >65	Any	Any	Unhealthy (Temperature)
50 – 65	<40 or >55	Any	Unhealthy (Humidity)

3. **Output Control:**

   - Display real-time data on the OLED screen of the node.
   - Green (Figure 1) and Red (Figure 2) LED bulbs indicate whether the predicted value is "Healthy" or "Unhealthy." The LED indicators model a scenario in which a connection to the bio-sauna is made to automatically adjust the temperature based on ML model predictions.
   - The mobile app also shows the real-time data with the predicted result.

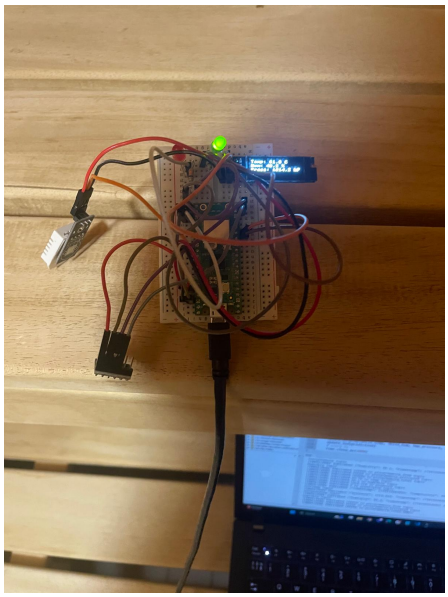


Figure 1: Green LED is on in the Sauna

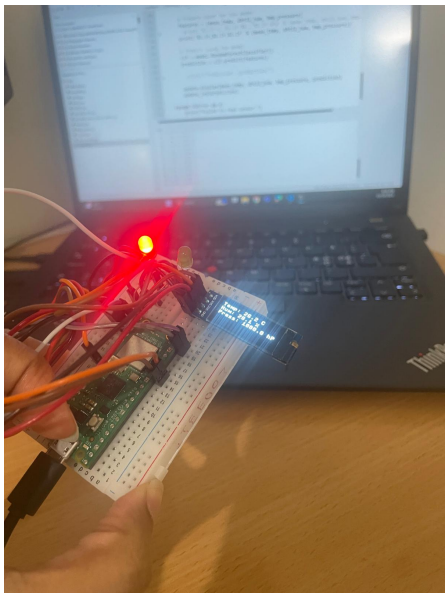


Figure 2: Red LED is on at room temperature

2 Additional Functionality

1. **Wireless Connectivity:**
   We established a wireless connection from the Raspberry Pi Pico W to a Wi-Fi network using MQTT since it is a lightweight publish/subscribe protocol. The Wi-Fi-enabled Pico W collected data from multiple sensors, including humidity, pressure, and temperature readings, and publishes this information to a MQTT broker, HiveMQ.

2. **Cloud integration:**
   The data received by HiveMQ is processed through Node-RED, which is a flow-based programming tool that we use to create logic for real-time monitoring and control. From Node-RED, the sensor data are routed to be stored in InfluxDB, a time-series database. This allows us to remotely monitor sauna conditions through a mobile app and powers LED comfort indicators in the environment.

3. **Mobile App:**
   We developed an Android application with MIT app inventor to display real-time data visualization and control using HiveMQ for sending and receiving data.

4. **Edge Machine Learning:**
   We implemented an edge-based machine learning model on the Pico W using everywhereml.

3 Architecture

The system contains layers of data communication as the sensing layer, networking layer, and data management layer. Figure 3 shows a block diagram of the system with the components inside a single sauna. The same set of devices are installed in other saunas to collect data.

3.1 Sensing Layer

The sensor layer contains the components below.

1. Raspberry Pi Pico W - Processes all the interfaces of the sensors and integrated with machine learning model. The machine learning model analyzes real-time sensor data to classify the sauna environment as "Healthy" or "Unhealthy."

2. BMP 280 - Collects real-time temperature and pressure data.

3. DHT 22 - Collects real-time humidity and temperature.

After the data is processed in the board, data is sent to the network layer using the MQTT protocol.

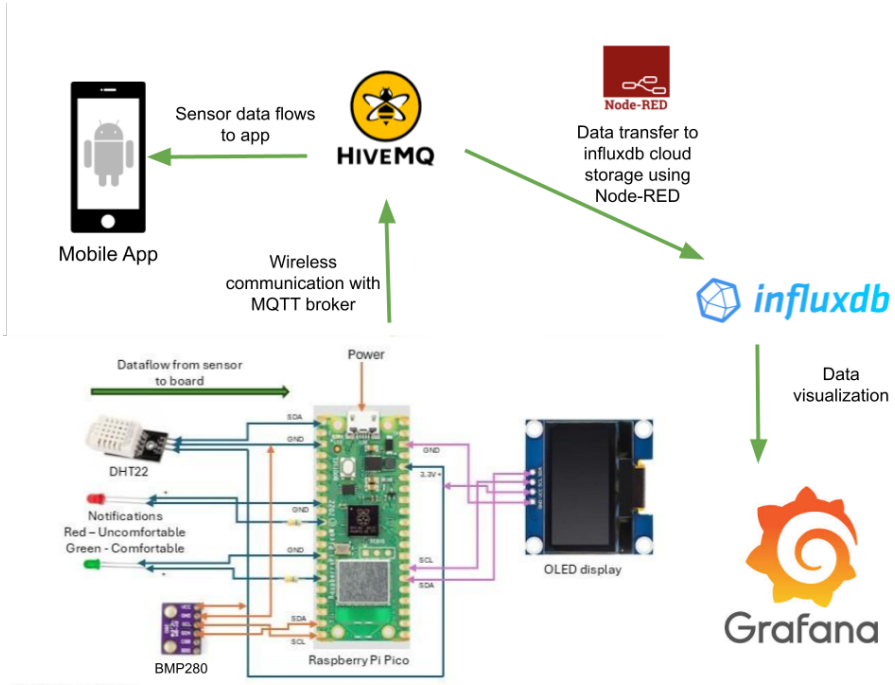


Figure 3: System Architecture

3.2 Network Layer

The purpose of this layer is to establish communication between the sensor layer and other downstream components. The layer contains:

1. MQTT protocol - This is a publish/subscribe protocol which helps to establish the connectivity between the client and the server.
2. Wi-Fi module - Enables the board to connect to Wi-Fi then to the MQTT broker.
3. MQTT broker - The broker facilitates the layer between the sensor layer and the data-management layer. It supports real-time communication with multiple devices, sensors, and clients. For this project we have used **hiveMQ** as the service provider.

3.3 Data Management Layer

In this layer, the data is stored, processed, visualized and managed to send alerts to the end user depending on the sensor data.

1. Data storage - Sensor data is stored in **InfluxDB** cloud service. It logs historical data for analysis and also for training machine learning models.

2. Data Visualization - For the data visualization part, we used **Grafana** as the platform. Here, the real-time data will be visualized in the dashboard.
3. Alert mechanism - The alert mechanism is established in three different ways.
   (a) LED bulbs: Green (healthy) and red (unhealthy) alerts on site.
   (b) OLED Display: Prediction result in display of the device.
   (c) Indicator in the mobile app: The real-time data and prediction of each sauna can be observed remotely.

4 Methods & Tools

4.1 Tools

1. Sensors: BMP280, DHT22
2. Devices/Output: Raspberry Pi Pico W, OLED, LED
3. Libraries: bmp280.py, umqtt.simple, everywhereml, network, socket,
4. Programming tools: MicroPython, Node-RED, MIT app inventor
5. Services: HiveMQ, InfluxDB
6. Visualization: Grafana

4.2 Methodology

4.3 Wireless Connectivity and Data Storage

For wireless connectivity, data storage, and visualization, the course exercises were used as the basis of our methods. MicroPython firmware was installed on the Raspberry Pi Pico W.

1. **Wireless Data Transfer:** The code we used connects a Raspberry Pi Pico W to a Wi-Fi network, reads environmental data from BMP280 and DHT22 sensors, and publishes the collected sensor data to an MQTT broker. We used libraries listed above for network connectivity, I2C communication, sensor interaction, and MQTT messaging. Credentials from the config file are accessed to connect to the specified Wi-Fi network and the MQTT broker, HiveMQ. The bmp280 and dht22 libraries are used to initialize the sensor and retrieve temperature and pressure readings. These readings are published to the HiveMQ every two seconds.
2. **Data Storage and Visualization:** Using Node-RED, the messages are sent to a bucket in InfluxDB along with a timestamp and the calculated latency. Node-RED listens for messages from HiveMQ and formats them using javascript to be sent to the specified bucket using an output node.

4.3.1 Machine Leaning Model Implementation

1. **Dataset Preparation:** The dataset was loaded and preprocessed to ensure randomness and consistency. The data was shuffled to prevent bias and reset to ensure a clean structure for further processing. This step ensures the training and testing processes are not influenced by the order of the data in the original dataset.

2. **Simulating Sensor Variability:** To mimic real-world sensor behavior, artificial noise was added into the dataset. This involved adding small random variations to the features: temperature, humidity, and pressure, simulating the inaccuracies and fluctuations commonly found in sensor data. This step makes the model more robust and better suited for real-world applications.

3. **Feature Selection and Target Variable:** The dataset was divided into two parts:

   - *Features:* The input variables (e.g., temperature, humidity, and pressure) used for making predictions.
   - *Target Variable:* The output label represents the prediction value.

   This separation helps the model focus on learning relationships between input features and their corresponding labels.

4. **Train-Test Split:** The data was split into training and testing sets. The training set was used to train the model, while the testing set was reserved for evaluating its performance. To split the data, train-test-split was imported from sklearn in Python.

5. **Data Normalization:** The feature values were standardized to have a mean of zero and a standard deviation of one. Normalization ensures that all input features are on the same scale, preventing features with larger ranges from dominating the learning process and improving the efficiency and accuracy of the model. The StandardScaler from sklearn was used for that.

6. **Model Training:** A Random Forest Classifier from everywhereml was used as the machine learning model [2]. This is an ensemble learning method that combines multiple decision trees to make predictions. By training the model on the prepared dataset, it learned to classify or predict based on patterns in the input features.

7. **Predictions:** The trained model was used to make predictions on new input data. Before making predictions, the input data was normalized using the same scaling process applied during training to maintain consistency. This demonstrates the model's ability to generalize and make accurate predictions for new cases.

8. **Saving the Model to Raspberry Pi Pico:** The trained model was exported to a format compatible with MicroPython, making it suitable for deployment on Raspberry Pi Pico. This step bridges the gap between training the model on a full-scale computer and using it in embedded systems. This allows the model to be reused for future predictions in real-time via the Raspberry Pi Pico.

4.3.2 Mobile app development

The mobile app is developed using the free tool 'MIT App Inventor' [1]. It's a high-level block-based visual programming language maintained by the Massachusetts Institute of Technology.

1. **Creating the interface layout**

   After initiating a new project in the MIT app inventor, the first step of the app development is to create the UI layout using the blocks given by the platform. Additional extensions can be installed under the "extensions" tab. For this project, we have used the AI2 MQTT Extension [6] developed by Zuletzt geändert. The configurations of the MQTT broker should be filled accordingly.

   The image 4 shows the app inventor interface where the home page is created.

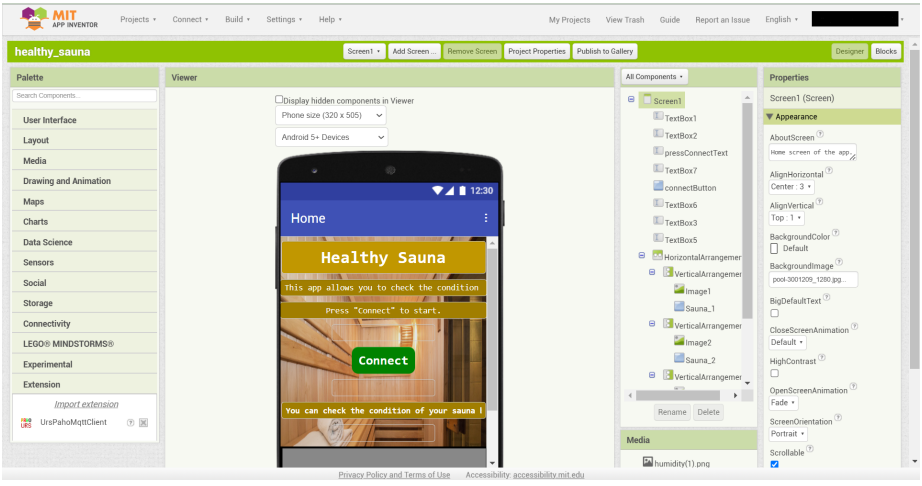


Figure 4: GUI layout of home page

2. **Creating the workflow**

   The function of the app is defined using the blocks appear in the top right corner of the tool. The blocks can be arranged according to the way

that the system should function. The image 5 shows the block arrangement of the home page. Various options for blocks can be selected from the "blocks" tab on the left.

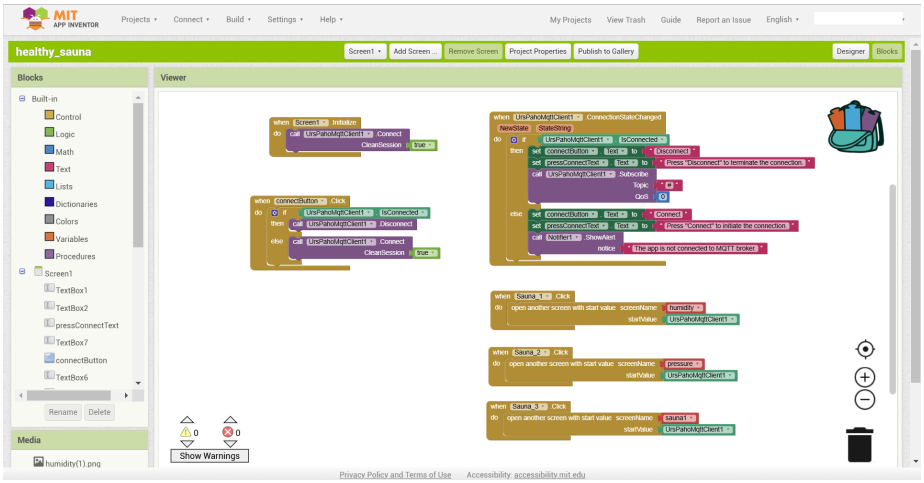
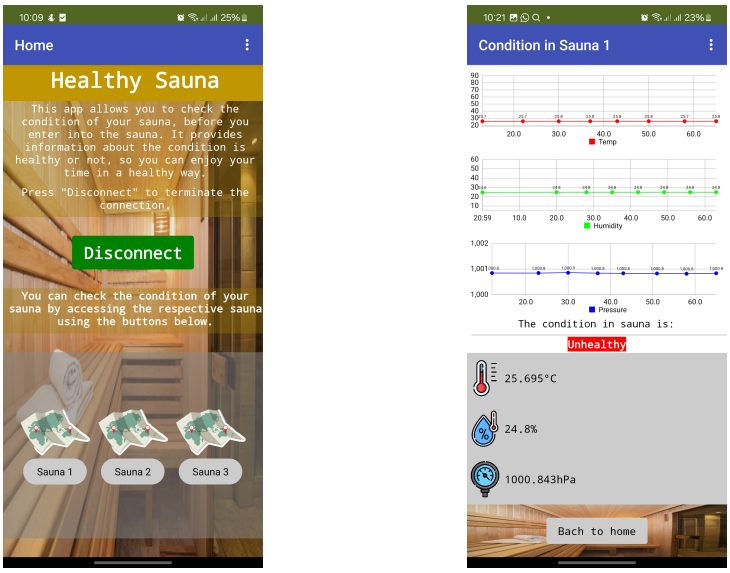


Figure 5: Blocks arrangement of home page

3. **Mobile app**

The mobile app can be generated by MIT app inventor easily by selecting the **build** option and then **Android App (.apk)**. An .apk file is generated and this file can be used to install the app on mobile. Figure 6a and 6b shows how the home page and data inside a sample sauna look in the app. The data was collected in room temperature, hence the values are lower than the values inside a sauna and the condition is "Unhealthy".



(a) Home page

(b) UI of Sauna 1

Figure 6: Mobile UI of the Healthy Sauna app

The X-axis shows the index of the values and Y-axis shows the temperature, humidity and pressure values accordingly. The exact value is shown as a label of each point.

5 Evaluation

5.1 System Performance Evaluation

For the first evaluation scenario we examined the performance of the system under two different conditions: regular room temperature (approximately 20°C) and elevated temperature inside a sauna (approximately 60°C). The goal of this evaluation scenario was to determine wether the high temperature of the sauna would impact the performance of the system.

For the evaluation criteria we chose to inspect changes in system latency and memory usage to measure any potential performance changes under the aforementioned conditions. The system was tested for 15 minutes under normal room temperature conditions and 20 minutes in the sauna condition at higher temperatures.

According to the Raspberry Pi Pico manual, the maximum operating temperature of the Pico is 85°C. Additionally, as noted in [[4]], the Pi's Component Operating Temperature is a combination of the Ambient Temperature and the Load-Induced Temperature Rise. Given the sauna's approximate temperature

of 60°C, it felt appropriate to investigate whether such conditions would lead to noticeable performance drops.

5.2 Machine Learning Model Evaluation

Evaluating a machine learning model can be done using various metrics and visualizations to understand how well the model performs.

5.2.1 Classification Report

The classification report provides a summary of key evaluation metrics for a classification model, broken down for each class. It includes:

- Precision: The proportion of true positive predictions out of all positive predictions. High precision means fewer false positives.
- Recall (Sensitivity): The proportion of true positive cases correctly identified out of all actual positives. High recall means fewer false negatives.
- F1-Score: The harmonic mean of precision and recall, balancing the two when there is an uneven class distribution.
- Support: The number of true instances of each class in the dataset.

In Python, classification-report from sklearn can be used to generate a classification report.

5.2.2 Confusion Matrix

The confusion matrix complements the classification report by giving a detailed breakdown of true positives, true negatives, false positives, and false negatives. It helps visualize where the model makes errors, such as misclassifying one class as another. This can be useful for debugging specific patterns of mistakes. For example, in binary classification, high false positives might indicate the model is over-predicting the positive class. The confusion matrix is a tabular representation of the model's predictions compared to the actual labels. It provides counts for:

- True Positives (TP): Correctly predicted positive instances.
- True Negatives (TN): Correctly predicted negative instances.
- False Positives (FP): Negative instances incorrectly predicted as positive.
- False Negatives (FN): Positive instances incorrectly predicted as negative.

In Python, we can generate and visualize the confusion matrix using confusion-matrix, ConfusionMatrixDisplay imports from sklearn.

5.2.3 Cross-Validation

Cross-validation is a method used to evaluate a model's generalizability by splitting the dataset into multiple parts (or folds). The model is trained on some folds and validated on others in rotation. This ensures the model's performance isn't overly dependent on a specific train-test split. The most common type is k-fold cross-validation, where the dataset is divided into k parts, and each part is used once as a validation set. In Python, we can perform cross-validation using cross-val-score from sklearn.

5.2.4 Learning Curve

A learning curve shows how the model's performance changes with the size of the training dataset. It helps diagnose problems like:

- Underfitting: Both training and validation scores are low.
- Overfitting: The training score is high, but the validation score is low.

The curve provides insight into whether adding more training data or improving the model complexity would enhance performance. In Python, we can generate a learning curve using learning-curve from sklearn.

5.2.5 Precision and Recall Curve

Visualizing the trade-off between precision and recall using a precision-recall curve gives a deeper understanding of the model's behavior across different thresholds. In Python, we can obtain precision recall curve and average precision score using precision-recall-curve, average-precision-score from sklearn.

5.2.6 ROC Curve and AUC

The ROC curve (Receiver Operating Characteristic) and AUC (Area Under the Curve) provide another way to evaluate the model, focusing on its ability to distinguish between classes. The ROC curve plots the true positive rate (sensitivity) against the false positive rate, while the AUC summarizes this as a single score. A higher AUC indicates better overall performance, with 1.0 being perfect and 0.5 indicating random guessing. In Python, we can obtain ROC curve, AUC using roc-curve, auc from sklearn.

Overall, these metrics and tools provide a comprehensive evaluation framework, helping identify strengths and weaknesses in the model and guiding improvements for better performance.

6 Results

6.1 Latency and Memory usage

To obtain the latency a calculation was performed: A timestamp was added to the message when it was sent from the Raspberry Pi, a second timestamp

Figure 7: Latency and temperature in room temp

Figure 8: Memory usage in room temp

was recorded when the message was received and processed by a Node-RED function. The latency was calculated by subtracting the first timestamp from the second. Memory usage was retrieved using Python module gc - garbage collection interface. The information collected is visualized using Grafana.

As shown in Figure 7 the latency in room temperature is around 4-5 seconds. Figure 8 displays that in room temperature the PI memory usage is 53kB, with 136kB of free space.

Figures 9 and 9 display similar readings as the room temperature graphs,

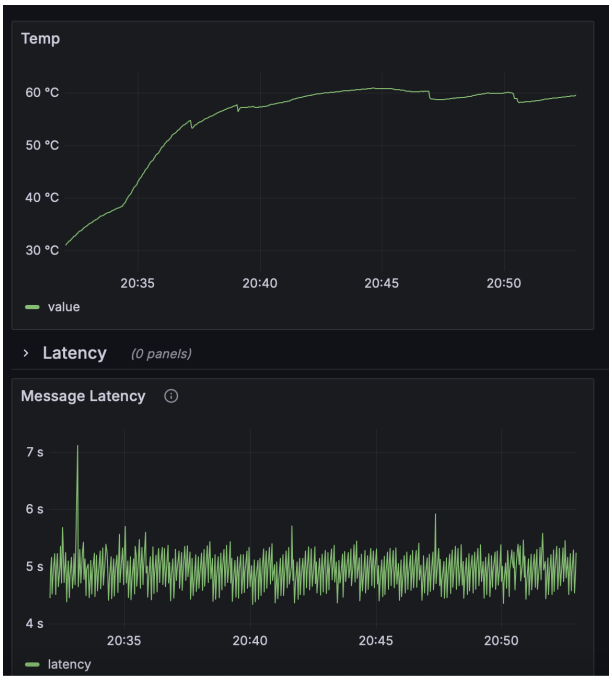


Figure 9: Latency and temperature in sauna

Figure 10: Memory usage in sauna

with latency being around 4-5 seconds and memory usage being 53kB. Based on these figures there are no notable changes in latency and memory usage when the system is in room temperature versus high-sauna temperature.

6.2 Results of Machine Learning Model Evaluation

6.2.1 Classification Report

The classification report in the Table 2 shows that the model performs exceptionally well, achieving an overall accuracy of 96%. For class 0 (unhealthy condition), the model has a precision of 93%, a recall of 99%, and an F1-score of 96%, indicating it is highly effective at detecting instances of this class with very few false negatives. Similarly, for class 1 (healthy condition), the model achieves a precision of 98%, a recall of 93%, and an F1-score of 95%, reflecting strong performance with minimal false positives. The macro and weighted averages for precision, recall, and F1-score are all 96%, demonstrating consistent performance across both classes. While the model is well-balanced, improving the recall for class 1 could further enhance its ability to correctly identify all instances of that class.

Table 2: Classification Report Results

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.93	0.99	0.96	67
1	0.98	0.93	0.95	67
Accuracy			0.96	134
Macro Avg	0.96	0.96	0.96	134
Weighted Avg	0.96	0.96	0.96	134

6.2.2 Confusion Matrix

The confusion matrix shows in Figure 11 provides a detailed breakdown of the model's performance by showing the number of correct and incorrect predictions for each class.

- True Positives (Class 0): 66 instances of class 0 were correctly predicted as class 0.
- False Positives (Class 1 misclassified as Class 0): Only 1 instance of class 1 was incorrectly predicted as class 0.
- False Negatives (Class 0 misclassified as Class 1): 5 instances of class 0 were incorrectly predicted as class 1.
- True Positives (Class 1): 62 instances of class 1 were correctly predicted as class 1.

Overall, the confusion matrix indicates strong performance, with minimal misclassifications. The model accurately predicts most instances for both classes, and the misclassifications are few, supporting the high precision, recall, and accuracy values observed in the classification report.

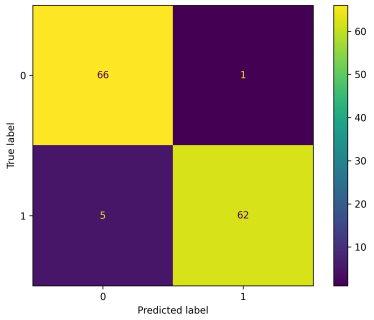


Figure 11: Confusion Matrix

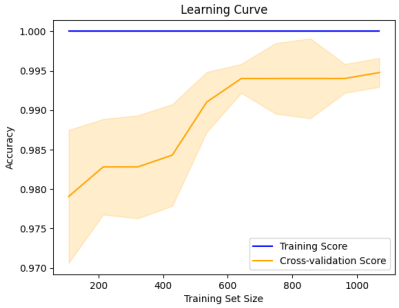


Figure 12: Learning Curve

6.2.3 Cross-Validation

The cross-validation scores indicate the model's performance across multiple splits of the dataset. We used 10 folds and the individual scores are [1.0, 0.99173554, 0.99173554, 0.99173554, 0.99166667, 1.0, 1.0, 0.99166667, 0.98333333 1.0], showing consistently high accuracy values across all folds, with minimal variability. This suggests that the model performs well on different subsets of the data, demonstrating strong generalization capabilities.

The mean cross-validation accuracy is 0.994, which confirms that the model achieves near-perfect accuracy on average. Such high accuracy indicates that the model is robust and reliable, with no significant overfitting or underfitting observed during cross-validation.

6.2.4 Learning Curve

The learning curve shown in Figure 12 illustrates the model's performance on both the training and cross-validation datasets as the training set size increases. The training score (blue line) remains consistently high, close to 1.0, indicating that the model perfectly fits the training data. However, the cross-validation score (orange line) starts around 0.98 and gradually improves as the training set size increases, showing that the model generalizes better with more data. The shaded region around the cross-validation score represents variability, and as the data size grows, this variability decreases, indicating more stable performance.

6.2.5 Precision and Recall Curve

The Precision-Recall (PR) curve shown in Figure 13 demonstrates the trade-off between precision and recall across different thresholds for the model's predictions. Precision refers to the proportion of correctly predicted positive instances, while recall measures the ability to capture all actual positive instances. In this plot, the curve stays close to 1.0 precision for most recall values, indicating that the model performs exceptionally well in classifying the positive class. The sharp

drop towards the end occurs when recall is maximized, but precision decreases due to a higher number of false positives at lower thresholds.

The Average Precision (AP) score, shown as 0.994, summarizes the area under the PR curve. This near-perfect score confirms that the model maintains a high precision and recall balance. Overall, the curve and AP score indicate that the model effectively handles the positive class, with minimal misclassifications and excellent predictive performance.

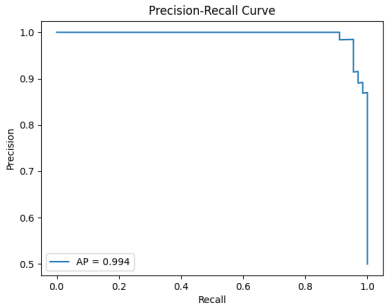


Figure 13: Precision and Recall Curve

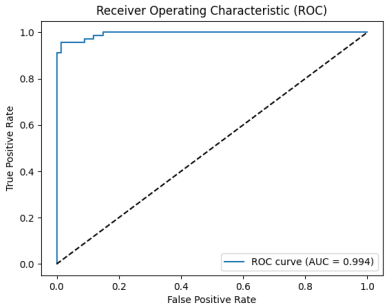


Figure 14: ROC Curve and AUC

6.2.6 ROC Curve and AUC

In the given ROC curve in Figure 14, the blue line represents the model's performance, while the diagonal dashed line indicates a random classifier with no discrimination ability (AUC = 0.5). The Area Under the Curve (AUC) for this model is 0.994, which is very close to the maximum possible value of 1. This demonstrates that the model has excellent classification capability, effectively separating positive and negative classes. The high AUC value indicates that the model achieves a strong balance between sensitivity and specificity, minimizing both false positives and false negatives.

7 Discussion

There were no noticeable changes in latency and memory usage when comparing the results of the room temperature and sauna temperature measurements. This indicates that the system is potentially able to handle temperatures up to 60°C. This aligns with the manufacturers specifications which state a maximum operating temperature of 85°C. However, to get more accurate results a duration longer than 20 minutes in the sauna conditions might have been required. The 20 minute period might not have allowed for the heat to accumulate to levels that could cause issues such as throttling.

For our purposes the 20 minute test period felt satisfactory because we wanted to avoid any equipment damage during our testing. Additionally, a

person would usually spend around 20 minutes in a sauna so this felt like a good and safe starting point for evaluating the system.

The memory usage of the system is good. There is still plenty of available memory to be used. The latency using mobile phone hotspot is around 4 seconds in both of the evaluation scenarios. There could be some room for improvement, but considering the use case of the system a 4 second delay can be tolerated as immediate response is not required for the system to function as intended. Using another network connection we have also been able to achieve a latency of 300-600 ms in room temperature conditions. These values are better than the values we got during our evaluation.

We suspect the main reason for this is that using different network connections as well as different providers can have significant impact on latency. Different connections can introduce additional hops to data packets that are being sent and received.

If the system is to be implemented to saunas with higher temperature, the devices must be capable of performing at higher level of temperature.

References

[1] MIT App Inventor — appinventor.mit.edu. `https://appinventor.mit.edu/`. [Accessed 18-12-2024].

[2] Eloquent Arduino. Micropython and machine learning: An introduction, 2024. Accessed: 18-12-2024.

[3] Fischer Barometer. Sauna: What should you consider in terms of humidity and air temperature?, 2024. Accessed: 18-12-2024.

[4] Brainboxes. Raspberry pi overheating guide. Accessed: 2024-12-18.

[5] Minna L. Hannuksela and Samer Ellahham. Benefits and risks of sauna bathing. *The American Journal of Medicine*, 110(2):118–126, 2001.

[6] Ulli. Ullis Roboter Seite/AI2 MQTT — ullisroboterseite.de. `https://ullisroboterseite.de/android-AI2-PahoMQTT-en.html`. [Accessed 18-12-2024].