UNIVERSITÀ DI PARMA

UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXXVII Ciclo

TinyRBF: On-Device Learning for Sensor Self-Calibration in Precision Agriculture Applications

Coordinatore:

Chiar.mo Prof. Marco Locatelli

Tutore:

Chiar.mo Prof. Michele Amoretti

Dottorando: *Francesco Saccani*

Anni Accademici 2021/2022 - 2023/2024

I would like to thank my supervisor, Professor Michele Amoretti, for offering me constant support and thought-provoking insights. A heartfelt thank you for always inspiring me with great passion for our work.

I am deeply grateful to Professor Stefano Caselli, for being an indispensable point of reference throughout the entire research period, always showing availability and professionalism.

A big thank you also goes to my company tutor Danilo Pau and the colleagues I met during my time at STMicroelectronics for their support and the interest they showed in me and our project.

I would like to thank my family for always supporting my choices and encouraging me throughout my academic journey.

I am deeply grateful to my girlfriend Annalisa for accompanying me with enthusiasm and patience on this path, proving to be a true model of determination and resilience for me.

A big thank you also goes to my colleagues Gabriele and Ernesto for always being there, through good and bad times, becoming brave companions on this adventure.

Table of Contents

List of Figures

List of Tables

Introduction

Precision agriculture leverages advanced technologies to optimize crop management, improve yields, and minimize environmental impact. Among the various technologies employed, the use of sensors is one of the most important practices, as they provide punctual data on parameters such as soil moisture, temperature, relative humidity, atmospheric pressure, and nutrient levels. These sensors enable real-time decision making and precise resource allocation. However, their accuracy is critical to maintaining the reliability of such systems. Over time, sensors experience drift, a phenomenon in which measurements deviate from the ground truth due to stresses such as thermal cycling, mechanical stress, electrical fluctuations, and aging. In the context of precision agriculture, this drift can lead to errors in irrigation scheduling, fertilizer application, and climate monitoring, ultimately impacting quality and productivity.

This Thesis adopts a use-case oriented approach, focusing on the practical challenges that have formed the basis for the development and evaluation of the proposed innovative solutions. In particular, the proposed solutions address the following three applications.

In-Vivo Plant Monitoring *Organic ElectroChemical Transistor* (OECT) devices, particularly the *Bioristor* sensor, are capable of determining the ionic content of biological systems and liquid samples. OECTs achieve this through an in-vivo and non-invasive installation directly within the plant stem, enabling the collection of real-time electrical data. A dedicated mathematical model correlates the sensor's electrical signals and the system's geometric parameters

to vital plant health indicators, such as ion concentration and water saturation levels. To enable efficiency and real-time decision making, this model should be solved directly in the control device to which the sensor is connected.

Continuous calibration of pressure sensors Pressure sensors are particularly difficult to compensate due to the structure of the sensing element which can change irreversibly when thermal or mechanical stresses are applied. During manufacturing, calibration typically relies on ground truth references, but these are available only briefly, during sample batch characterization and validation. A common method, *one-point calibration*, involves comparing the sensor's readings to a highly accurate reference, calculating a single offset, and programming it into the sensor. However, this approach is inadequate for most real-world applications. A three-phase experiment was designed to simulate the complete life cycle of a sensor: from manufacturing, where a first calibration and network initialization takes place; to motherboard soldering, where thermal and mechanical stresses introduced during assembly must be taken into account; up to end-user usage, where recalibration is infrequent. Four different datasets covering the LPS22HH and LPS22DF barometric pressure sensors were used to investigate both normal and extreme operating conditions.

In-field correction of environmental data This use case addresses the challenges of maintaining accurate sensor measurements in the harsh conditions of field installations. Outdoor environmental sensors are exposed to fluctuating temperatures, humidity, and other stresses that can cause measurement drift and inaccuracies over time. The experimental approach involves comparing data from six different sensors to reference measurements provided by a high-precision weather station. The objective is to develop and validate algorithms that can correct sensor outputs in real time, ensuring reliable and consistent data collection despite environmental stresses.

The focus in this Thesis is on bringing computational solutions as close as possible to edge devices, with the objective of minimizing latency and enhancing the relevance and accuracy of data processing. This approach also reduces reliance on

third-party cloud platforms, ensuring greater data privacy and ownership, key concerns in many modern applications. Such an edge-centric strategy necessitates the adoption of on-device learning techniques, enabling the devices to continuously adapt and improve based on real-time data. However, implementing these methods poses significant challenges due to the inherent computational limitations of edge devices. These constraints are particularly pronounced in agriculture, where nodes must be low-cost and energy-efficient to remain viable. Moreover, in agricultural contexts, edge nodes are often battery-operated and expected to function autonomously for the entire crop growth season. This imposes strict requirements on power consumption, making the design of computationally lightweight and resource-aware algorithms an essential part of the solution. By overcoming these challenges, this Thesis contributes to advancing practical, scalable, and efficient on-device learning methods for real-world agricultural applications and beyond.

For the first use case, a comprehensive research was conducted to identify the most efficient technique to solve the mathematical model within the constraints of an edge computing environment. The experimental evaluation examined the quality of the model solution, focusing on its accuracy and reliability, while also analyzing the memory usage to ensure efficient use of limited resources on embedded devices. Additionally, the execution time was measured to verify that the solution could meet the needs of real-time applications. Considering the edge computing context, the solution was implemented and deployed on resource-constrained embedded devices to analyze the computational complexity and overall energy consumption. While the adopted approach demonstrated the effectiveness and efficiency of employing AI in resource-constrained devices, the current solution is inherently supervised. It relies on a pre-collected dataset obtained from monitoring multiple plants under water stress conditions. Consequently, the model is static once deployed on the final device, lacking the ability to evolve over time or adapt to specific applications and changing conditions. This limitation serves as the foundation for the solution proposed for the second and third use cases.

The challenges and state-of-the-art approaches for on-device learning in devices with extremely limited resources were explored in coping with the second and third

use cases. Recently proposed alternatives to the back-propagation learning scheme, such as the Forward-Forward and PEPITA algorithms, were analyzed in detail. Considering the limitations of the approaches in literature, a new solution based on Radial Basis Function (RBF) networks was proposed. The proposed model, named *TinyRBF*, is a Gaussian RBF network with the ability to modify its structure over time to find the right trade-off between prediction accuracy and time and memory complexity.

Chapter 1

Context and Motivation

1.1 Precision Agriculture

Precision agriculture (PA), also known as precision farming or smart farming, is an innovative approach that integrates advanced technologies to optimize the use of agricultural resources, enhance productivity, and minimize environmental impact [1]. With the evolution of technology and the increasing availability of new tools and methodologies, PA has gained significant attention in recent decades, becoming a cornerstone for sustainable agriculture and efficient resource management [2, 3, 4].

PA coincides with a set of techniques that use data collection and analysis technologies to monitor and manage variability within a field. The primary goal is to maximize yield per unit of area while minimizing the use of input resources (such as water, fertilizers, and pesticides), thus reducing costs and environmental impact [5]. PA aims to achieve high production efficiency, targeted resource management, and improved product quality.

Among the key technologies that underpin PA, Global Positioning Systems (GPS) and Geographic Information Systems (GIS) are essential for mapping fields, monitoring crop variability, and optimizing in-field treatments. These tools allow precise tracking of agricultural machines and accurate distribution of inputs [6].

The use of satellite imagery and drones equipped with multispectral sensors is

another key component of PA. These devices allow for the monitoring of crop health, irrigation problems, fertilization issues, and pest infestations in real-time. The images collected from drones or satellites are analyzed to generate crop vigor maps that display areas of water stress, nutrient deficiency, or disease damage [4, 7].

Internet of Things (IoT) enables the integration of sensors in the field that collect real-time data on variables such as temperature, humidity, soil pH, wind speed, and light levels. These data can be used to optimize irrigation, fertilization, and other agricultural practices [8].

Furthermore, big data analysis combined with artificial intelligence (AI) and machine learning provides tools to identify hidden patterns, predict crop yields, and optimize resource use. Deep learning techniques, in particular, process images from drones and satellites to improve diagnostics [9].

Last but not least, autonomous agricultural machines, including tractors, planters, and harvesters, can perform operations such as planting, harvesting, and crop management autonomously. These vehicles, equipped with GPS and sensors, can operate precisely and without human intervention, reducing the risk of errors and increasing efficiency [10].

Applications of PA span irrigation management, precision fertilization, pest and disease control, crop health monitoring, and yield prediction. The use of soil moisture sensors and weather stations allows for real-time monitoring of soil moisture conditions and precise determination of when and how much to irrigate, reducing water wastage and improving water use efficiency [11]. Data collected from soil sensors and drones allow fertilizers to be applied precisely where needed. This reduces fertilizer consumption, costs, and the environmental impact from excess use, such as the risk of water eutrophication. The use of drones equipped with sensors to detect early signs of diseases and pest infestations allows farmers to intervene promptly with localized treatments, reducing pesticide use and improving crop health [12]. Remote sensing technologies and satellite imagery can detect water stress, nutrient deficiencies, or infestations, enabling farmers to make informed decisions to optimize agronomic practices.

Despite its advantages, PA faces challenges such as high initial costs for acquiring

advanced technologies, the need for technical expertise, and difficulties in integrating various systems into cohesive platforms [1]. Limited internet connectivity in rural areas also poses obstacles to adopting remote monitoring and data-sharing solutions. Nevertheless, the future of PA looks promising with emerging technologies such as artificial intelligence, machine learning, and open, long-range, low-power communication protocols, such as LoRaWAN [13, 14]. These innovations are expected to offer more precise and connected agricultural management. Open-source technologies and collaborative efforts among agricultural enterprises may reduce costs and make PA solutions more accessible, fostering sustainable agricultural practices globally.

1.1.1 Sensors in Precision Agriculture

Sensors are a cornerstone of PA, enabling real-time data collection and informed decision-making for optimizing resource use and improving crop yields. Among the various sensor types, soil moisture sensors play a critical role in irrigation management by providing insights into the water content of the soil. Accurate soil moisture data ensure that crops receive the optimal amount of water, reducing water waste and preventing issues such as over-irrigation or drought stress [15].

Soil moisture sensors have revolutionized irrigation practices in PA by enabling site-specific water management. They reduce water consumption, promote sustainable farming, and prevent crop damage caused by inconsistent irrigation. Integration with IoT networks and remote monitoring systems enhances the scalability and usability of these sensors, making them vital components of modern agricultural systems [16].

Despite their utility, soil moisture sensors face challenges that can impact the reliability of their measurements. Among these, measurement drift is one of the most critical issues [17]. Measurement drift refers to the gradual deviation of a sensor's output from its true value over time, which can result from different factors:

- Environmental Factors: Variations in temperature, humidity, and soil composition can introduce drift in sensor readings. Capacitance sensors, for instance, are sensitive to changes in temperature and soil salinity, which can alter the

dielectric properties of the soil and lead to inaccurate moisture readings.

- Aging and Wear: Over time, the electronic components of soil moisture sensors, such as electrodes or semiconductors, degrade, reducing their precision. Corrosion or fouling caused by prolonged exposure to soil and water further exacerbates this issue.
- Soil Heterogeneity: Non-uniform soil conditions, such as variations in texture, organic matter, or compaction, can cause discrepancies in sensor measurements, leading to apparent drift when the sensors are moved to different locations.

An example of measurement drift is illustrated in Figure 1.1. The figure shows data from two soil moisture sensors installed at different depths. Over time, a noticeable decrease in the measured values and their range can be observed. Given the inherent complexity of the soil-plant-atmosphere continuum and the ongoing irrigation and precipitation processes, it is unclear whether this decline is an actual decrease over time of soil moisture, or rather sensor's response is also affected by gradual wear of its electronic components. In the latter case, drift could be mitigated through the implementation of an appropriate calibration model to provide the user with more accurate measurements.

Another challenge associated with soil moisture sensors is the interpretability of their measurements. Even when two sensors are installed under identical conditions and in the same position, they can produce differing readings, as demonstrated in the graph in Figure 1.2. In such cases, determining the precise soil moisture value becomes challenging. Instead, attention is often focused on monitoring fluctuations in moisture levels over time, which can provide valuable insights into the water stress state of a plant.

While soil moisture sensors are indispensable tools in precision irrigation, addressing challenges like measurement drift is crucial for maintaining their effectiveness. By combining recalibration with advanced data processing, these challenges can be mitigated, ensuring accurate soil moisture monitoring and sustainable agricultural practices. However, the recalibration process is not only time-consuming but

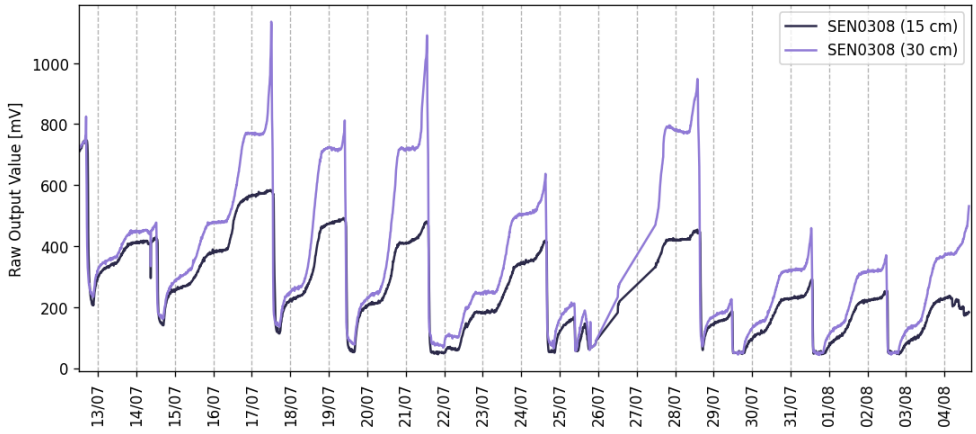


Figure 1.1: Output values of two soil moisture sensors affected by measurement drift.

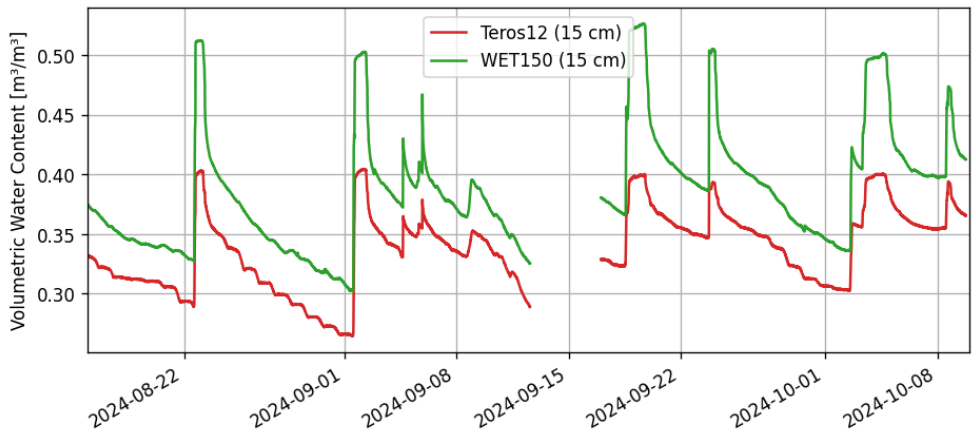


Figure 1.2: Volumetric water content measured by two different sensors installed under identical conditions and in the same position.

also resource-intensive, often requiring specialized equipment and expertise. This challenge becomes particularly pronounced in large-scale farming, where recalibrating a network of sensors distributed across vast areas introduces significant logistical difficulties.

Additionally, recalibration relies on the availability of a reference value, or ground truth, to adjust and validate sensor readings. For soil moisture sensors, obtaining this ground truth is particularly challenging. It typically involves installing a high-precision sensor, such as a gravimetric moisture meter or a laboratory-calibrated probe, in conditions identical to those of the sensor being recalibrated. Achieving identical environmental conditions, including soil composition, compaction, and moisture distribution, is inherently complex and may not be feasible in every location. Moreover, high-precision sensors are often prohibitively expensive, making their widespread deployment impractical for many farming operations.

1.2 Use Cases

This Thesis adopts a use-case-oriented approach, emphasizing real-world scenarios to ground its exploration and evaluation of innovative solutions. By focusing on practical challenges, the research ensures that the proposed solutions are not only theoretically robust but also applicable in real-world contexts. Specifically, the Thesis targets three key application areas, each representing a distinct set of challenges and opportunities. These applications serve as the foundation for assessing the effectiveness, efficiency, and impact of the developed solutions, demonstrating their relevance and potential in addressing practical needs.

1.2.1 In-Vivo Plant Monitoring

From applications to hardware platforms, edge computing is drastically changing the computing landscape [18, 19]. A new frontier in Smart Agriculture [20, 21, 22] is combining edge computing to enable on-field raw data processing [23, 24, 25] with nanobiotechnology to enable smart plant sensors that transmit plant chemical signals to on-field agricultural and phenotyping equipment [26, 27, 28, 29].

Devices called *Organic ElectroChemical Transistors* (OECTs) are used to determine the ionic content of biological systems and liquid samples. The current I that gradually changes from zero (no flux) to a steady state value of current, which is proportional to the concentration and properties of ions in solution, is the signal that is measured by an OECT. The relationship between I and the properties of ions, such as concentration, charge, and size, can be understood using mathematical models [30, 31].

A mathematical model that explains the behavior of OECTs in which the geometry of the system's internal fluidic circuits can alter over time was proposed by Gentile et al. [32]. Ion transport is made possible by these circuits, which are a system of chambers and tubes that allow liquid solution to move from the gate to the drain-source electrodes. The mathematical model calculates the system's ion concentration using the data produced by the OECTs. Specifically, this tool is now being utilized to evaluate the data generated by the *Bioristor* device, developed by IMEM-CNR [26, 28].

The objective of this use case is to introduce an enhanced version of the mathematical model that is both more accurate and easier to solve. Additionally, it aims to present and evaluate several algorithms for numerically solving this model, including brute force, adaptive methods, Newton's method, gradient descent, and multilayer perceptron. These algorithm implementations need to be designed to be energy-efficient and optimized for real-time execution on resource-constrained edge devices. Experimental results should cover the quality of the model solution, memory usage, execution time and energy consumption.

1.2.2 Continuous Calibration of Pressure Sensors

Over time, sensors tend to deviate from the ground truth or expected values due to various stresses, including thermal, mechanical, electrical, and aging effects. These subtle errors accumulate and compromise the accuracy of measurements. While end users purchase sensors expecting reliable performance, they are often unaware of these errors, which ideally should be automatically and continuously corrected during operation.

Pressure sensors in particular pose significant challenges. Thermal and mechanical stresses can induce irreversible changes in the 3D lattice structure of the sensor chip, resulting in time-varying hysteresis and nonlinear drifts. These drifts are caused by factors such as chip soldering, temperature fluctuations, and prolonged exposure to high temperatures. As a result, pressure sensors require recalibration throughout their lifetime.

During manufacturing, calibration is typically based on ground truth references, but these are only available for a limited period of time, during sample batch characterization and validation. A common method, single-point calibration, involves comparing sensor readings to a highly accurate reference, calculating a single offset, and programming it into the sensor. At runtime, this offset is applied to correct sensor readings. However, this approach is inadequate for most real-world applications.

Artificial Intelligence (AI) offers a promising solution for drift compensation. However, implementing AI-based methods inside a sensor chip requires minimal complexity, low power consumption (e.g., nanowatt levels), and limited resources (e.g., 0.5 KB of memory).

A key challenge is the concept drift problem [33], where a model's performance degrades due to changes in the underlying data distribution or operating conditions. Since these changes cannot be predicted reliably offline, pre-trained AI models are unsuitable for such applications. Addressing concept drift requires adaptive solutions capable of learning in real-time to ensure consistent performance.

To meet these requirements, it is crucial to avoid traditional back-propagation algorithms [34], which are computationally intensive and unsuitable for resource-constrained devices. Instead, an online learning approach is necessary, enabling the sensor to update its model dynamically and incrementally during operation [35], without relying on extensive computational resources. Such an approach ensures real-time adaptability while adhering to the strict constraints of on-device learning.

A comprehensive three-phase experimental framework was designed to simulate the complete lifecycle of a atmospheric pressure sensor, including the critical stages from manufacturing to end-user operation.

Sensor Manufacturing Phase 1 corresponds to the mass production stage of the

sensors and represents the initial calibration and model initialization process. During this phase, the sensors are produced in a controlled factory environment, ensuring consistent manufacturing quality.

A small random sample of sensors is selected from each production batch for performance verification. These sensors undergo thermal stress simulations, replicating the soldering process typically encountered during integration into end-user devices. This procedure evaluates the sensors' response under such conditions, identifying performance variations or deviations that could inform adjustments to the calibration process.

A global model is initialized during this phase with the objective of minimizing the overall error across all sensors. The model is designed to be as general as possible, ensuring that it captures common patterns and characteristics shared by all devices in the batch. This allows the same global model to be deployed uniformly across all sensors, avoiding any need for device-specific specialization. The approach simplifies initial deployment while establishing a strong baseline for subsequent adaptation or refinement in later phases.

Motherboard Soldering Phase 2 replicates the conditions experienced by a sensor during and immediately after it is soldered onto the motherboard of the final device. This phase accounts for the thermal and mechanical stresses introduced during the assembly process, which can significantly impact the sensor's performance and calibration. By simulating these conditions, Phase 2 aims to assess and mitigate any sensor drift or degradation resulting from this critical step. Specifically, this phase simulates the soldering process, during which sensors are exposed to significant thermal stress.

Unlike Phase 1, where a global model was deployed uniformly across all sensors, Phase 2 involves further specialization of the global model. The model is incrementally adapted to account for the unique drift behavior of each individual sensor. This ensures that the calibration is tailored to compensate for sensor-specific deviations while maintaining overall performance.

The experimental setup for Phase 2 assumes a quasi-laboratory validation en-

vironment, where reference values are frequently available. These references provide a reliable basis for calibrating and fine-tuning the sensor models, simulating an ideal scenario where precise ground-truth measurements support real-time adaptation.

End-User Phase Phase 3 represents the operational life of the sensor during its deployment and use by the end user. This phase focuses on simulating real-world conditions, where the sensor operates over an extended period and is subject to environmental stresses, long-term drift, and infrequent recalibrations, reflecting typical challenges encountered in practical applications.

The primary objective in this phase is to establish a calibration process that seamlessly integrates into the user experience. This process must be adaptable to a wide range of operating conditions, from standard use to extreme stress scenarios, ensuring consistent and reliable sensor performance regardless of the environment or usage patterns.

A critical assumption during Phase 3 is that acquiring reference values for the sensor – necessary for calibration – is a resource-intensive process. As such, reference measurements are only available sporadically, requiring the calibration method to function effectively even with limited ground-truth data. This constraint underscores the importance of implementing robust, adaptive algorithms capable of compensating for drift and maintaining accuracy between calibration events, enabling reliable long-term operation with minimal user intervention.

Four separate datasets covering post-soldering, normal and thermal stressed operating conditions were used to evaluate the behavior of the sensor during these phases. These datasets cover two barometric pressure sensors, LPS22HH and LPS22DF, which have been analyzed to understand performance variations and drift characteristics under different stress factors.

1.2.3 On-Field Correction of Environmental Data

This use case addresses the challenges of maintaining accurate sensor measurements in outdoor field installations, where environmental conditions are unpredictable and often harsh. Outdoor environmental sensors play a vital role in applications such as weather monitoring, climate research, and precision agriculture, where data accuracy is critical for informed decision-making. However, these sensors are exposed to environmental stresses, including fluctuating temperatures, varying humidity levels, precipitation, and prolonged wear from sunlight and wind. Over time, these factors can lead to measurement drift and inaccuracies, compromising the reliability of the collected data.

To investigate and address these challenges, a sensor node was implemented and deployed in the field. This node incorporated six different sensors, each designed to measure environmental parameters such as temperature, relative humidity, and atmospheric pressure. The node was installed in an outdoor setting alongside a nearby high-accuracy weather station, which served as the reference system for validation. Over the course of nearly two months, data from the environmental node were collected and compared to the reference measurements provided by the weather station.

The comparison enabled the identification of deviations caused by environmental stresses, providing valuable insights into sensor drift and inaccuracies over time. This dataset served as the foundation for developing and testing adaptive algorithms capable of correcting sensor outputs in real time. The primary objective was to ensure that the corrected data remained consistent and reliable despite the harsh field conditions.

The algorithm was designed to operate efficiently on the sensor node itself or through lightweight computational frameworks, minimizing the need for high computational power or frequent manual intervention. This approach aimed to reduce or eliminate the need for frequent recalibration, which is often impractical for large-scale or remote deployments.

1.3 Challenges and Approaches

1.3.1 Use Case 1

The use case involving OECT devices for determining high-level plant characteristics, such as ion concentration and water saturation of biological systems, presents several challenges. These challenges arise from the need to process dynamic sensor data in real time, perform computations directly on resource-constrained edge devices, and balance the trade-off between computational complexity, solution accuracy and energy efficiency. To address these challenges, the research question can be formulated as follows: *"How can we design and implement an enhanced mathematical model for OECT data analysis that achieves accurate real-time processing, supports edge computing deployment, and optimizes the trade-off between computational complexity, solution accuracy and energy consumption on resource-constrained devices?"*.

The mathematical model and its numerical solution algorithms must be capable of processing dynamic OECT data streams in real time. Real-time data processing ensures timely insights, which are critical for applications such as biological monitoring or precision irrigation. The solution must be designed to handle streaming data efficiently without introducing latency that could compromise the utility of the results.

To minimize the reliance on cloud-based computation and reduce data transmission costs, the solution must be deployable on edge devices located close to the data source. These devices are inherently resource-constrained, with limited memory, processing power, and energy capacity. The resolution algorithms must therefore be lightweight and optimized for execution on such devices, such as microcontroller units (MCUs), without compromising the accuracy or robustness of the solution. The ability to perform computations directly on edge devices also enhances data privacy and system reliability, as it reduces dependency on external infrastructure.

Achieving a balance between computational complexity, error tolerance, and energy consumption is critical for the viability of the solution on edge devices. While highly accurate models may demand extensive computational resources, they can be

impractical for real-time execution on resource-limited platforms. Conversely, overly simplified models may introduce unacceptable levels of error. The enhanced mathematical model and its solving algorithms must therefore be designed to balance these factors effectively. The solution should minimize computational complexity while maintaining sufficient accuracy for practical applications and, at the same time, reduce energy consumption to enable prolonged operation of battery-powered devices.

The enhanced mathematical model and its associated resolution algorithms must address three key challenges: real-time data processing, deployment on edge devices, and managing the trade-offs between complexity, accuracy, and energy efficiency. Experimental evaluation should validate the solution by measuring its performance across several dimensions, including model accuracy, memory usage, execution time, and energy consumption. By meeting these requirements, the solution aims to enable robust, efficient, and practical OECT data analysis for biological and environmental applications.

1.3.2 Use Cases 2-3

Use Cases 2 and 3 were designed to address a critical research question: *"Is it possible to achieve accurate online incremental learning for a regression task without incurring catastrophic forgetting, and without relying on back-propagation algorithms, while ensuring deployability on highly resource-constrained devices (e.g., sensors) starting from no assigned budget?"*. To answer this question, any proposed or existing solution must satisfy several stringent requirements tailored to the unique challenges of real-time, on-device learning in constrained environments.

Incremental learning ensures that the neural model can continuously adapt and update its knowledge as new data become available [35]. This capability is essential for addressing concept drift [33], where the data distribution changes over time, requiring the model to adjust dynamically to maintain accuracy. A solution capable of incremental learning must avoid rigid training methods and embrace flexibility to integrate new information seamlessly while preserving relevant prior knowledge.

Online learning involves processing data sequentially as it arrives, in contrast to traditional batch learning or offline approaches that rely on having complete visibility

over the entire dataset [35]. This requirement eliminates the dependency upon storing large datasets, making it feasible for resource-constrained devices with limited memory. Furthermore, it ensures that learning is performed in real time, enabling immediate adaptation to evolving data streams, which is critical for real-world applications like environmental monitoring.

Back-propagation (BP) [34] is the standard de facto algorithm for parameter updates in neural networks. However, it is computationally expensive, requires significant power, and demands substantial memory for storing activations and datasets during training. These requirements make BP unsuitable for devices such as sensors and MCUs with limited computational and energy resources. Therefore, solutions must explore alternative learning paradigms that are lightweight, efficient, and capable of operating within stringent resource constraints.

The proposed solution must be deployable on tiny devices with severe hardware limitations. MCUs typically feature memory capacities ranging from tens of kilobytes to a few megabytes, while sensors often operate with even smaller memory footprints, such as a few tens of kilobytes. Any learning algorithm must operate within these constraints, ensuring that the solution is not only theoretically valid but also practical for deployment in real-world applications.

While Catastrophic Forgetting is traditionally seen as a limitation of neural networks, where the learning of new information leads to the loss of previously acquired knowledge [36], it also presents an opportunity. By strategically leveraging forgetting, models can manage complexity by dynamically pruning neurons or parameters that no longer contribute significantly to predictions. This approach allows the model to adapt to evolving data while maintaining manageable complexity and avoiding overfitting to outdated patterns, ensuring its accuracy over time.

These requirements collectively define the framework within which solutions for online incremental learning on resource-constrained devices must operate. The dual challenges of addressing concept drift and resource limitations necessitate innovative approaches that depart from traditional learning algorithms like back-propagation. By meeting these criteria, the proposed solution aims to enable robust, real-time learning on tiny devices, leading the way for their application in diverse fields such as

environmental monitoring, precision agriculture, and embedded AI.

Chapter 2

Supervised In-Sensor Data Analysis

This chapter investigates the integration of Organic Electrochemical Transistors (OECTs) with edge computing to enhance real-time monitoring of plant health in Smart Agriculture. Energy-efficient algorithms for interpreting OECT data on resource-constrained devices are presented. Specifically, the considered OECT data are ion concentration and water saturation in plant stems.

The structure of the chapter is as follows. Section 2.1 discusses related work in the context. The mathematical model describing the behavior of an OECT sensor is presented in Section 2.2. Section 2.3 details the algorithms designed to solve the model, while Section 2.4 focuses on the energy-efficient implementation of these algorithms targeting embedded systems.

2.1 Related Work

In a recent review [23], Qazi et al. noted that edge AI applications in smart agriculture are still emerging. These solutions are expected to equip farmers with real-time threat detection and timely prevention capabilities, although there is a trade-off between accuracy and power consumption.

Brunelli et al. [24] developed an AI-based embedded system for real-time pest detection, using computer vision algorithms. Specifically, they designed an IoT neural network trap targeting the codling moth, a harmful pest for apple crops. The system captures an image of the trap, pre-processes it, crops each insect for classification, and sends an alert to the farmer if a codling moth is detected. The system's classification accuracy, recall, and power consumption were evaluated. Although the proposed solution uses a Raspberry Pi 3 and Intel Movidius Neural Compute Stick — rather than ultra-low-power processors — the low duty cycle minimizes average power consumption.

Gia et al. [25] developed an experimental framework that implements CNN-based image compression directly within sensor nodes to accommodate the low data transmission rates of LoRa and NB-IoT wireless sensor networks. This advanced compression reduced data size by up to 67%, with a decompression error of less than 5%. The sensing nodes used an nRF24L01 module, an AVR ATmega8 microcontroller, and multiple sensors, with energy consumption measured at 138.92 mJ per minute using the MonSoon power monitor tool. An edge gateway was created with a Raspberry Pi 3 connected to an nRF2401+ module to receive data from the nodes. This module's long antenna improved reception but increased power consumption, so the gateway was connected to a power socket to address this issue.

2.2 Mathematical Model

OECT devices are biocompatible, minimally invasive sensors that enable real-time monitoring of ion concentration and water saturation within a plant stem. An OECT device consists of two textile electrodes inserted into the plant stem: one electrode, made from a conductive polymer (commonly PEDOT:PSS), is connected at its ends to the drain and source, functioning as the transistor channel; the other is a reference electrode connected to the gate, as illustrated in Figure 2.1.

The system's behavior is modeled in two dimensions, with the plant stem divided into N vessels. Each vessel represents a path that ions can travel through if immersed in the electrolyte (i.e., plant sap).

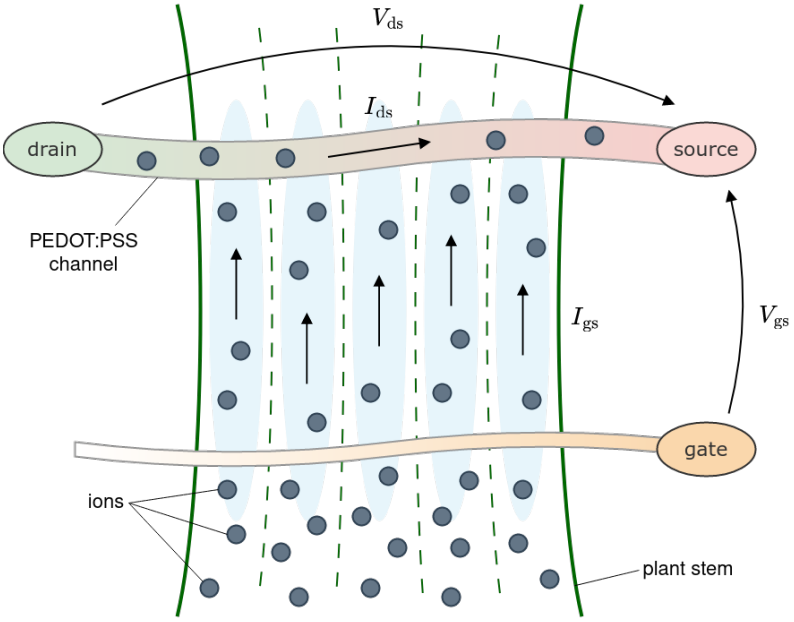


Figure 2.1: Schematics of an OECT device applied to a plant stem.

When a positive voltage $V_{ds,0}$ is applied between the drain (ground) and the source (V_{ds}), and between the gate and source (V_{gs}), positive ions in the electrolyte are driven into the channel and toward the source. This movement generates two currents: one between the gate and source (I_{gs}) and another between the drain and source (I_{ds}). Both currents depend on the system's geometric and physical properties, the ion concentration (C), and the water saturation level, represented as S, which indicates the fraction of plant vessels accessible to the ions.

The absolute value of the electric current I_{ds} varies between a minimum value I_{ds}^{on} when the gate is active (i.e., $V_{gs} = V_{gs,0}$) and a maximum value I_{ds}^{off} when $V_{gs} = 0$ V. When $V_{gs} = V_{gs,0}$, the ions are pushed into the channel, which reduces its conductivity; consequently, I_{ds} takes on its minimum value.

To determine the values of I_{ds}^{on} and I_{ds}^{off}, V_{gs} is defined as a square wave voltage:

$$V_{gs} = \frac{V_{gs,0}}{2}\left(1 + \text{sign}\left(\sin\left(\frac{2\pi t}{T}\right)\right)\right) \tag{2.1}$$

where t is the time and $T/2$ is the period that enables the current to transition to a steady state.

Consequently, in an OECT device, the currents I_{ds}^{on}, I_{ds}^{off}, and I_{gs}^{on} can be sampled once during each cycle, specifically at each rise and subsequent fall of the V_{ds} signal. Therefore, the system's frequency of response is $1/T$.

The mathematical model that relates all the parameters in the system and enables the transformation of the outputs from an OECT device into microscopic (C) and macroscopic (S) variables is derived in [32] and presented in Equation 2.2.

$$\begin{cases} I_{ds}^{on} = I_{gs}^{on} + \dfrac{V_{ds}}{R_{dry} + S\left(\frac{R_{wet}^{off}}{\text{mod}(C)+1} - R_{dry}\right)} & (2.2.1) \\ I_{ds}^{off} = \dfrac{V_{ds}}{R_{dry} + S\left(R_{wet}^{off} - R_{dry}\right)} & (2.2.2) \\ I_{gs}^{on} = \dfrac{V_{gs}}{R_{stem}(C)}S & (2.2.3) \end{cases}$$

The model is structured as a system of three equations involving various parameters and variables:

- C: ion concentration in the electrolyte, measured in molar concentration;
- S: water saturation in the plant stem, expressed as the fraction of vessels immersed in the electrolyte;
- I_{ds}^{on}: electric current between the drain and source when the gate is active, measured in Amperes;
- I_{ds}^{off}: electric current between the drain and source when the gate is inactive, measured in Amperes;
- I_{gs}^{on}: electric current between the gate and source when the gate is active, measured in Amperes;
- V_{ds}: electric voltage applied between the drain and source, measured in Volts
- V_{gs}: electric voltage applied between the gate and source, measured in Volts;
- R_{dry}: electrical resistance of the dry PEDOT:PSS channel before electrolyte exposure, measured in Ohms;
- R_{wet}^{off}: electrical resistance of the wet PEDOT:PSS channel after electrolyte exposure with the gate inactive, measured in Ohms;
- $\mathrm{mod}(C)$: modulation of the PEDOT:PSS channel, representing the system's gain in electric current relative to a reference, due to ion concentration C;
- $R_{stem}(C)$: electrical resistance of the plant stem as a function of ion concentration, measured in Ohms.

Here, R_{dry} and $R_{stem}(C)$ depend on the system's geometric and physical characteristics, while V_{ds} and V_{gs} are device inputs, and I_{ds}^{on}, I_{ds}^{off}, and I_{gs}^{on} are sensor outputs and the independent variables in the model. The dependent variables in the model are C, R_{wet}^{off}, and S.

The physical quantities $\mathrm{mod}(C)$ and $R_{stem}(C)$ are defined by the following functions of the ion concentration C:

$$\mathrm{mod}(C) = a \cdot \ln(C) + b \tag{2.3}$$

$$R_{stem}(C) = c + d \cdot C^{0.955} \tag{2.4}$$

where a, b, c and d are experimentally determined parameters. This formulation, as

opposed to the one presented in [32], was derived based on empirical data and is well-suited to the typical ion concentration range found in plant sap. An ad hoc experiment was conducted to determine the behavior and parameter values, using several Bioristor sensors fully immersed in aqueous solutions with various concentrations of KCl, ranging from 10^{-4} M to 10^{-1} M.

Starting from the mathematical model expressed as a system of three equations, it is possible to solve for S from Equation 2.2.3 and R from Equation 2.2.2, then substitute these into Equation 2.2.1 to derive a single equation that depends only on the ion concentration C, as shown in Eq. 2.5. This reformulation restricts the solution search domain solely to ion concentration values, allowing formulas (2.6) and (2.7) to be used to determine R_{wet}^{off} and S.

$$I_{gs}^{on} + \frac{V_{gs}\left[V_{ds}\left(I_{ds}^{off} - I_{ds}^{on} + I_{gs}^{on}\right) + I_{ds}^{off}\left(V_{ds} - I_{ds}^{on}R_{dry} + I_{gs}^{on}R_{dry}\right)\cdot \mathrm{mod(C)}\right]}{I_{ds}^{off}R_{dry}(I_{ds}^{on} - I_{gs}^{on})\cdot R_{stem}(C)\cdot \mathrm{mod}(C)} = 0 \tag{2.5}$$

$$R_{wet}^{off} = \frac{R_{dry}V_{ds}\left(I_{ds}^{off} - I_{ds}^{on} + I_{gs}^{on}\right)\cdot(\mathrm{mod}(C)+1)}{V_{ds}\left(I_{ds}^{off} - I_{ds}^{on} + I_{gs}^{on}\right) + I_{ds}^{off}\left(V_{ds} - I_{ds}^{on}R_{dry} + I_{gs}^{on}R_{dry}\right)\cdot \mathrm{mod}(C)} \tag{2.6}$$

$$S = \frac{V_{ds}\left(I_{ds}^{off} - I_{ds}^{on} + I_{gs}^{on}\right) + I_{ds}^{off}\left(V_{ds} - I_{ds}^{on}R_{dry} + I_{gs}^{on}R_{dry}\right)\cdot \mathrm{mod}(C)}{I_{ds}^{off}R_{dry}\left(I_{gs}^{on} - I_{ds}^{on}\right)\cdot \mathrm{mod}(C)} \tag{2.7}$$

Due to the specific characteristics of $\mathrm{mod}(C)$ and $R_{stem}(C)$, neither the system in Equation 2.2 nor the Equation 2.5 can be solved using an explicit formulation of their variables. Therefore, approximation algorithms are necessary to identify potential solutions to the mathematical model.

2.3 Algorithms

2.3.1 Brute Force

The brute force approach is a systematic method for evaluating a function at various input values that are evenly distributed within a specified range. Specifically, for a function $f(x)$, this method requires defining a minimum value x_{min}, a maximum value x_{max}, and the number of steps N to be taken. The algorithm begins by evaluating f at the starting point $x_0 = x_{min}$ and subsequently calculates the next inputs using the formula:

$$x_{i+1} = x_i + \frac{x_{max} - x_{min}}{N} \tag{2.8}$$

This process continues iteratively until x_i exceeds x_{max}. The algorithm identifies the solution as the input x_k that results in the smallest error, which is determined by the absolute value of the function at that point, $|f(x_k)|$.

2.3.2 Adaptive

The adaptive algorithm is comparable to the brute force method but employs a sparser distribution of values and iteratively narrows the search range to achieve a more accurate solution in less time. Specifically, for a given function $f(x)$, an initial search interval $[x_{min,0}, x_{max,0}]$, and a specified number of steps N, the algorithm first applies the brute force technique over the interval, retaining the best k solutions found. Next, it calculates the midpoint $x_{mean,0}$ and updates the bounds of the search range as follows:

$$x_{min,i+1} = \max\left(x_{mean,i} - \tfrac{x_{max,i} - x_{min,i}}{2} \cdot r, x_{min,0}\right) \tag{2.9}$$

$$x_{max,i+1} = \min\left(x_{mean,i} + \tfrac{x_{max,i} - x_{min,i}}{2} \cdot r, x_{max,0}\right) \tag{2.10}$$

Here, $0 < r < 1$ is the reduction factor that determines how much the interval is shrunk. This iterative process continues until either a predetermined maximum number of iterations is reached or the error falls below a specified tolerance threshold.

The final output of the algorithm is the midpoint obtained from the last iteration, representing an approximation of the optimal solution. This method enhances effi-

ciency by focusing on promising regions of the search space, allowing for quicker convergence compared to traditional brute force techniques.

2.3.3 Gradient Descent

The gradient descent algorithm, commonly known as the steepest descent method, is an optimization technique employed to locate the minima of a function. For a differentiable function f and an initial point x_0, the algorithm generates a sequence of progressively improved approximations $x_1, x_2, x_3, \ldots$ of the function's minimum. The first value in this sequence is computed using the formula:

$$x_1 = x_0 - \gamma_0 f'(x_0) \tag{2.11}$$

Here, f' denotes the first-order derivative of f, and γ_0 represents the initial learning rate or step size. For $i > 1$, the algorithm updates the sequence according to the rule:

$$x_i = x_{i-1} - \gamma_i f'(x_{i-1}) \tag{2.12}$$

In this case, γ_i is the learning rate, which is adjusted at each iteration using the Barzilai–Borwein gradient method:

$$\gamma_i = \frac{|(x_i - x_{i-1})\,[f'(x_i) - f'(x_{i-1})]|}{\|f'(x_i) - f'(x_{i-1})\|^2} \tag{2.13}$$

The fundamental concept behind this technique is to start from a random point on the curve of the function and iteratively move in the direction opposite to the gradient, as this direction reduces the function's value.

The algorithm concludes when either a maximum number of iterations is reached, the error falls below a specified tolerance threshold, or the gradient becomes negligibly small.

Since the goal is to find the roots of the function f rather than its minima, an alternative function $g(x) = f^2(x)$ is defined for which the gradient descent method is applied. In this way, $g'(x) = 2f(x)f'(x)$ and the minima of g correspond to the zeros of f.

2.3.4 Newton's Method

Newton's method, often referred to as the Newton-Raphson method, is an iterative root-finding algorithm designed to approximate the roots of a function. For a differentiable function f and an initial estimate x_0 for a root, the algorithm produces a sequence of approximations $x_1, x_2, x_3, ...$ using the following formula at each iteration i:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{2.14}$$

where f' denotes the first order derivative of f. The underlying principle of the method involves beginning with an initial guess x_0, constructing a tangent line to the function at this point, and using the x-intercept of that tangent line as the next approximation for the root. This process continues iteratively until convergence is achieved. Since the tangent line closely approximates the function near x_i, the x-intercept is expected to provide a better approximation of the root compared to x_i.

The algorithm concludes when either a specified maximum number of iterations is reached, the error falls below a designated tolerance threshold, or the gradient becomes sufficiently small.

2.3.5 Multilayer Perceptron

A MultiLayer Perceptron (MLP) is a type of feedforward artificial neural network characterized by a minimum of three fully connected layers: an input layer, one or more hidden layers, and an output layer. The nodes in successive layers are interconnected through a linear transformation represented by the equation:

$$y = \mathbf{W} \cdot x + \mathbf{b} \tag{2.15}$$

In this equation, $\mathbf{W}$ denotes the weight matrix, while $\mathbf{b}$ represents the bias vector. All nodes, except those in the input layer, function as neurons that apply a non-linear activation function, which can include options such as the sigmoid, hyperbolic tangent (tanh), or rectified linear unit (ReLU) functions.

In this case study, a neural network was utilized to tackle a regression task aimed at mapping the output currents from a specific Bioristor sensor to values of ion

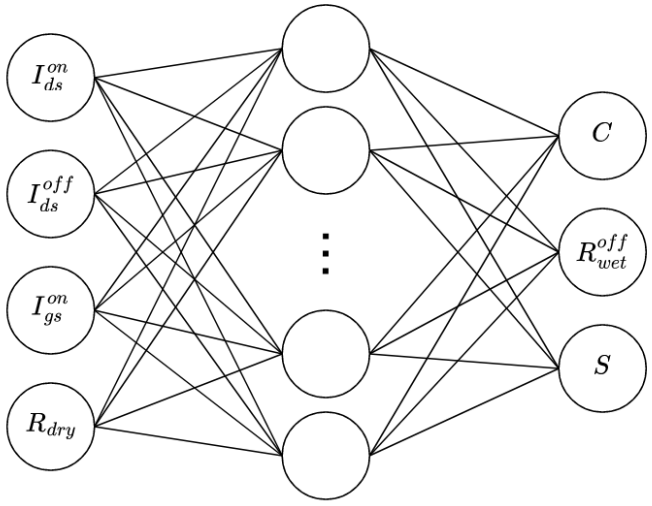


Figure 2.2: Generic topology of the implemented neural network.

concentration, wet channel resistance, and water saturation. Consequently, the constructed network features four inputs (I_{ds}^{on}, I_{ds}^{off}, I_{gs}^{on} and R_{dry}), one or more hidden layers, and three outputs (C, R_{wet}^{off} and S), as illustrated in Figure 2.2.

2.4 Implementation

The mathematical model and algorithms were implemented using Rust [37], a relatively new system programming language developed by the Mozilla Foundation. Rust is designed to achieve execution performance comparable to that of C and C++, while ensuring safety without relying on a runtime or garbage collector. Its rich type system and ownership model provide guarantees for memory and thread safety, helping to prevent various types of bugs at compile time. Rust's performance, reliability, and set of utility tools have contributed to its adoption across various application domains, particularly as a strong candidate for embedded devices [38]. This programming language employs compile-time abstractions that do not impose runtime penalties for

safety checks, resulting in smaller programs, reduced CPU operations, and consequently, lower power consumption and memory usage on hardware.

The Rust programming language was utilized to implement a Command-Line Interface (CLI) for rapid prototyping and model validation, as well as an embedded library targeting ARM Cortex-M processors. This library is publicly available on the GitHub platform[1].

A specific Rust compiler option, known as *opt-level*, was the focus of this study. This setting determines the number of optimizations applied by the compiler to the code, with values ranging from 0 to 3. Higher optimization levels can yield faster runtime performance, although they may result in longer compilation times. An alternative option, “s", is designed to optimize for binary size. Each experiment was conducted using both the value 3 and “s" to examine the differences between these two build configurations.

[1] `https://github.com/dsg-unipr/bioristor-lib`

Chapter 3

On-Device Learning for Sensor Calibration

Radial Basis Function (RBF) neural networks have demonstrated considerable effectiveness in tasks requiring precise function approximation and classification. However, limitations in learning efficiency, model complexity, and task adaptability often restrict their use in large-scale applications. This chapter introduces a novel RBF-based neural network model, named Tiny Radial Basis Function (TinyRBF), designed to address these limitations through a refined topology, efficient learning process, and optimized adaptation strategies.

The chapter begins with Section 3.1 providing an overview of state-of-the-art approaches and related work in the field of On-Device Learning (ODL), as well as alternative neural network models with adaptive structures and efficient inference mechanisms. This section examines existing methods for dynamic neuron allocation, pruning strategies, and initialization techniques, highlighting the strengths and limitations of these approaches to position the proposed model within the current research landscape.

The network's topology is then presented in Section 3.2, describing its structure and distinctive features, including the dynamic number of neurons in the hidden layer. The inference process is outlined in Section 3.3, while the learning process is detailed

in Section 3.4 with a focus on adaptations that enhance convergence and accuracy, as well as the pruning strategy applied to maintain controlled model complexity over time. An overview of the evolution of the on-device model is depicted in Section 3.5. Section 3.6 explores the optional initialization strategy, designed to ensure stability and performance by carefully setting the centers and widths of the RBF units. Finally, Section 3.7 provides an analysis of time and memory complexity, evaluating the model's computational demands.

3.1 Related Work

On-Device Learning (ODL) has gained significant attention in recent years, driven by the growing need for intelligent systems that can adapt in real time in resource-constrained environments such as IoT devices and edge computing. Unlike traditional centralized training, ODL emphasizes models and algorithms designed to perform incremental updates or online learning directly on embedded systems with limited computation, memory, and energy resources.

This section reviews major advances in the field, structured into three main categories:

1. *Reformable TinyML*: Subsection 3.1.1 explores techniques aimed at dynamically adapting tiny machine learning (TinyML) models for on-device learning. These methods focus on maintaining high efficiency and low resource consumption, while allowing the model to evolve and effectively handle new data.

2. *Forward alternatives to backpropagation (BP)*: Subsection 3.1.2 reviews emerging algorithms that replace traditional BP with forward-only learning approaches. These methods eliminate the need for computationally expensive backward steps, making them well-suited for inference-optimized hardware and resource-constrained settings.

3. *Different Neural Network approaches*: Subsection 3.1.3 highlights several neural network architectures, such as Restrict Coulomb Energy and Radial Ba-

sis Function networks. These approaches leverage their unique properties to achieve fast learning, robustness, and low computational overhead.

3.1.1 Reformable TinyML

Building on the TinyML paradigm – which focuses on deploying models for inference on inexpensive, low-power devices – various approaches have been proposed to address the high memory and computational demands of the backpropagation (BP) algorithm [39]. To handle concept drift, these methods aim to make models adaptable through local updates or Over-The-Air (OTA) updates, enabling *reformable* online learning [40]. However, while many of these systems operate with relatively relaxed device constraints, algorithms designed for more resource-constrained environments often rely on fine-tuning the initial model. This limits the potential for fully autonomous online learning [41].

3.1.2 Forward Alternatives to Backpropagation

"Forward-only" algorithms have emerged as novel learning approaches that train neural networks using only forward passes, bypassing the backward pass integral to the traditional backpropagation algorithm. These methods take advantage of the capabilities of modern neural processing units, which are highly optimized for inference tasks, enabling efficient and high-performance execution. By eliminating the need for back-propagating the error, forward-only schemes can simplify the computational requirements and potentially unlock new hardware optimizations tailored to forward inference dynamics, making them attractive for both research and practical applications in resource-constrained environments.

The *Forward-Forward* (FF) algorithm, introduced by Hinton in [42], is an alternative to backpropagation for training neural networks. Instead of computing gradients through a backward pass, it uses two forward passes: one with "positive" data (real examples) and another with "negative" data (synthetic or contrasting examples). The objective of learning is to maximize a *goodness* measure for positive data and minimize it for negative data. While many formulations of this measure are possi-

ble, the paper proposed to use the sum of squared neural activities as the goodness measure for a layer. Some methods for generating negative data have been proposed, for instance, in case of an image classification task, part of the image can be used to encode the label, where some pixel contain the one of N representation of the label. In this way, mislabeled inputs can be intuitively used as false data. However a general approach is missing. In particular for regression tasks, there is no straightforward way to create corresponding negated examples. Furthermore, two ways of performing inference are described. The first consists in providing a neutral input (i.e., without indicating the label) and using the activations of the various layers as input to a previously trained softmax for classification. Otherwise, it is necessary to perform a forward pass for each output class with the corresponding label encoded in the input, and choose as output prediction the one that generated the highest goodness. The downside is that the first procedure produce sub-optimal results, while the second one is not scalable in terms of complexity.

The *Predictive Forward-Forward* (PFF) algorithm [43] is a learning framework that extends the FF algorithm, focusing on biological plausibility and computational efficiency. It integrates the principles of predictive coding and eliminates the need for backpropagation. The PFF system consists of two interconnected circuits:

1. *Representation Circuit*: This circuit learns distributed representations of input data. It evaluates goodness functions locally at each level during forward-only processing. Learning occurs via iterative updates based on goodness scores, facilitating classification tasks.
2. *Generative Circuit*: This circuit works in conjunction with the representation circuit, learning a direct top-down generative model. It predicts neural activity in lower levels, enabling error-driven Hebbian-like updates. This design supports better interpretability and negative sample synthesis, which are critical for training.

The *μ-FF* algorithm [44] is another variant of Hinton's Forward-Forward framework, built on the core idea of FF, but designed to optimize the algorithm for resource-constrained environments such as microcontrollers. The μ-FF algorithm structures the learning task into two main components: the first component focuses on feature ex-

traction, while the second component performs the training by solving a multivariate Ridge regression problem. This approach allows for obtaining a closed-form solution to the problem, eliminating the need for gradient descent or any other iterative optimization methods.

However, neither PFF nor μ-FF solve the limitations highlighted above that characterize the FF algorithm, and that make it unsuitable for the discussed applications.

PEPITA (Present the Error to Perturb the Input To Modulate the Activity) [45] is another forward-only learning method. It operates in two stages: the *standard pass*, where the output error is computed through a conventional forward pass, and the *modulated pass*, where the input is adjusted based on the error of the first pass, and the resulting activities are used to update the network parameters. The error is applied to the input shape via a random matrix F with zero mean and small standard deviation, enabling the input to be perturbed in a way that influences the network's learning. Considering a neural network composed of L layers, the activation values of the standard and modulated passes are computed as in Equations 3.1 and 3.2, respectively.

$$h_1 = \sigma_1(W_1 x), h_l = \sigma_l(W_l h_{l-1}) \quad 2 \leq l \leq L \tag{3.1}$$

$$h_1^{\text{err}} = \sigma_1(W_1(x + Fe)), h_l^{\text{err}} = \sigma_l(W_l h_{l-1}^{\text{err}}) \quad 2 \leq l \leq L \tag{3.2}$$

Here, σ_l represents the non-linear activation function of layer l, W_l denotes the weight matrix between layers $l-1$ and l, and e is the output error of the network in the standard pass. After the two forward passes, network weights for the first, intermediate and output layers are updated using the formulas in Equation 3.3.

$$\begin{aligned} \Delta W_1 &= (h_1 - h_1^{\text{err}}) \cdot (x + Fe)^\top \\ \Delta W_l &= (h_l - h_l^{\text{err}}) \cdot (h_{l-1}^{\text{err}})^\top \quad 2 \leq l \leq L \\ \Delta W_L &= e \cdot (h_{L-1}^{\text{err}})^\top \end{aligned} \tag{3.3}$$

It is clear that a notable drawback of PEPITA is the need to store activations from the standard pass in memory, as the parameter updates in the modulated pass rely on these activations. This introduces memory requirements similar to those of traditional backpropagation, reducing its efficiency in memory-constrained environments. In addition, the work by Srinivasan et al. [46] demonstrated that PEPITA is a specific case

of the Forward-Forward algorithm. While FF requires a distinct process to generate negative data, PEPITA directly incorporates error information into the original input through feedback mechanisms.

A variant to PEPITA algorithm, namely *MEMPEPITA* (MPE) [47], addresses the memory-related problem performing a further third pass in parallel. The latter consists of a process similar to the modulated pass that uses sensor data as input and computes the activations only at the time they are needed. In this way, MPE drastically reduces memory usage while increases by one third the complexity of the learning process.

On the other hand, *PEPITA-time-local* (PTL) [46] proposes an approximation of the PEPITA learning rule that determines two different parameter updates applied during the standard and modulated forward passes, respectively.

$$\begin{aligned} \Delta W_l &= h_l h_{l-1}^{\text{err}\top} - h_l^{\text{err}} h_{l-1}^{\text{err}\top} \\ &\approx h_l h_{l-1}^{\top} - h_l^{\text{err}} h_{l-1}^{\text{err}\top} \end{aligned} \tag{3.4}$$

The approximation, reported in Equation 3.4, ensures that past activity information is not required and, therefore, there is no need to store in memory the activation values of the standard pass. However, this approach introduces a slightly negative impact on the model accuracy.

In the paper by Kohan et al. [48], a novel learning framework called *Signal Propagation* (SP) is introduced. This framework employs two concurrent forward passes: one for processing sensor data and another for a learning signal. The learning signal is initially generated by a target generator, which adjusts the output target to match the input's dimensionality, and subsequently follows the same process as the sensor data in the forward pass. At each layer, a local loss is computed based on the activations from both the input data and the generated target. This loss is then used to update the parameters, and the output, along with the target, is forwarded to the next layer. However, a key limitation of the SP framework is that the definition of the target generator function is not clearly established and is highly task-dependent. Despite this, several use cases and examples are provided in the paper.

3.1.3 Different Neural Network Approaches

Restricted Coulomb Energy (RCE) networks, introduced in [49], are hyperspherical classifiers with three layers: input, hidden, and output. The input layer is fully connected to the hidden layer, while each hidden neuron is linked to a single output node, representing the label for classification. Hidden neurons are dynamically initialized hyperspheres, each defined by a center point (a sample from the feature space) and a radius. The learning process of an RCE network involves the following steps:

- allocating a new neuron if an input does not fit within existing hyperspheres;
- reducing the radius of neurons responsible for misclassification to exclude the misclassified input;
- identifying and removing neurons with a small radius, as they are considered redundant.

A recent variant, called *TinyRCE*, was introduced in [50]. This approach integrates a feature extractor, such as a convolutional neural network trained offline via backpropagation or randomly initialized as in Extreme Learning Machines (ELM), to generate features for the TinyRCE classifier. To address memory saturation caused by adding new neurons, a culling mechanism was introduced, based on two attributes for each neuron:

1. *age*: incremented every time the neuron activates;
2. *reliability*: adjusted based on the correctness of the neuron's classifications.

Neurons with low firing frequency and poor reliability are identified as redundant and removed. Additionally, the mechanism for creating new neurons was refined to prevent hypersphere overlap. Noise resilience was improved by considering as outliers any incorrect predictions from neurons with reliability values in the lowest quartile among all hyperspheres. While RCE networks are inherently unsuitable for regression tasks, their mechanism of dynamically adapting hidden neurons can inspire models capable of addressing complex tasks in an online, adaptable manner.

A *Radial Basis Function* (RBF) network [51] is a feed-forward neural network consisting of three layers:

1. input layer: receives I input features and pass them directly to each hidden neuron;

2. hidden layer: composed of N RBF units;
3. output layer: dense connection between hidden unit activations and O outputs of the network.

Each hidden neuron k uses a radial basis function $\theta_k(\boldsymbol{x})$, typically a Gaussian function, as its nonlinear activation function:

$$\theta_k(\boldsymbol{x}) = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{c}_k\|^2}{r_k^2}} \tag{3.5}$$

Here, $k \in \{1,...,N\}$, $\|\cdot\|$ represents the Euclidean norm, and $\boldsymbol{c}_k$ and r_k denote the center and standard deviation of the Gaussian function, respectively. The hidden layer nodes are fully connected to the output layer, and the network outputs are computed as:

$$y_i(\boldsymbol{x}) = \omega_{0,i} + \sum_{k=1}^{N} \omega_{k,i}\theta_k(\boldsymbol{x}) \tag{3.6}$$

where $i \in \{1,...,O\}$, $k \in \{1,...,N\}$, $\omega_{k,i}$ represents the weight from the k-th hidden neuron to the i-th output node, and $\omega_{0,i}$ is the bias weight. The learning process in an RBF network involves determining the centers $\boldsymbol{c_k}$, the connection weights $\omega_{k,i}$, and the standard deviations r_k. Initially, the centers and radii are selected using methods such as random sampling from the training data or clustering algorithms. Subsequently, the network weights are optimized using linear techniques like least squares. While several variations of this learning process have been proposed [51], they typically rely on batch learning, where the full training dataset or a significant portion of it is used during training.

The *Resource Allocating Network* (RAN) [52] is a sequential learning algorithm designed to construct Gaussian RBF networks incrementally in an online manner. Starting with no hidden neurons, the network adds a new node whenever both the prediction error and the distance to the nearest center exceed predefined thresholds. Additional mechanisms have been incorporated to limit the frequency of node allocations, update the network when new units are not added, and enhance resilience to noise.

While the RAN algorithm includes a threshold mechanism for adding new neurons, several later studies introduced pruning strategies to prevent unchecked network

growth. In [53], the authors proposed the *Minimal RAN* (MRAN) algorithm, which uses neuron activation values as the basis for pruning. Specifically, hidden units are removed if their activation remains below a defined threshold for a specified number of consecutive learning steps. This is achieved by assigning a counter to each RBF neuron to track consecutive threshold violations. However, this approach uses constant values for allocation thresholds which may be unsuitable for input signals that change over time.

The *General Growing And Pruning RBF* (GGAP-RBF) network [54, 55] introduced the concept of *significance* for hidden neurons. This measures the information a neuron contributes to the function being learned, representing its average contribution to the network output across all input data. A new neuron is added if its significance exceeds a predefined learning error, and it is removed if its value falls below the target precision. To reduce computational complexity, the authors proposed that parameter adjustments occur only for the neuron closest (in Euclidean distance) to the most recent input, provided no new neuron is added. Similarly, during pruning, only the most recently adjusted neuron is checked for removal. However, calculating significance assumes prior knowledge of the input feature distribution, which is often unrealistic in scenarios like sensor data.

There have been attempts to combine RCE and RBF mechanisms, such as [56] which, however, does not describe the training process used for the reported function approximation results. Several enhancements to the original RAN algorithm and pruning strategies have been proposed [51, 57], but many do not consider the stringent computational constraints typical of microcontroller units (MCUs) and sensor-based systems.

RBF networks have demonstrated performance comparable to multilayer perceptron (MLP) models while offering significantly faster training speeds [51]. This efficiency makes them particularly well-suited for on-device incremental and online learning tasks, where computational resources and energy are often limited. Unlike MLPs, which rely on gradient-based optimization requiring multiple iterations, RBF networks utilize localized activation functions and straightforward learning processes, enabling faster convergence. These characteristics position RBF networks as

an ideal choice for scenarios such as edge computing, real-time decision-making, and adaptive systems in resource-constrained environments like IoT devices, sensors, and embedded systems.

3.2 Topology

The proposed on-device online learning solution draws inspiration from the RAN neuron allocation process [52] and the pruning mechanism introduced by MRAN [53]. Both of these approaches laid the foundation for adaptive and dynamic neuron allocation in RBF networks, ensuring that the model grows and prunes neurons based on the complexity of the input data. However, the current approach introduces several key improvements to further improve the adaptability and efficiency of the network.

First, the method has been improved to better adapt to changes in the input data over time, allowing the network to adapt its structure more effectively in response to evolving patterns or distributions in the real-time data. This is especially important for applications where the input data may fluctuate or shift and a static network structure would fail to maintain optimal performance.

Second, the hyperparameter selection process has been simplified, making the algorithm easier to configure and more accessible for practical implementation in on-device learning scenarios.

These innovations make the proposed solution more robust, adaptable, and practical for implementation in dynamic, real-time environments, where data distribution can continuously change.

More specifically, the proposed solution is a Gaussian RBF network with the ability to modify its structure over time to find the right compromise between prediction accuracy and time and memory complexity. The model is an artificial neural network composed of three layers: input, hidden and output. Figure 3.1 reports an example of network topology with 2 inputs and 1 output.

The input layer plays the role of the entry point for data into the network and directly passes the input features into the hidden layer without applying any transformation.

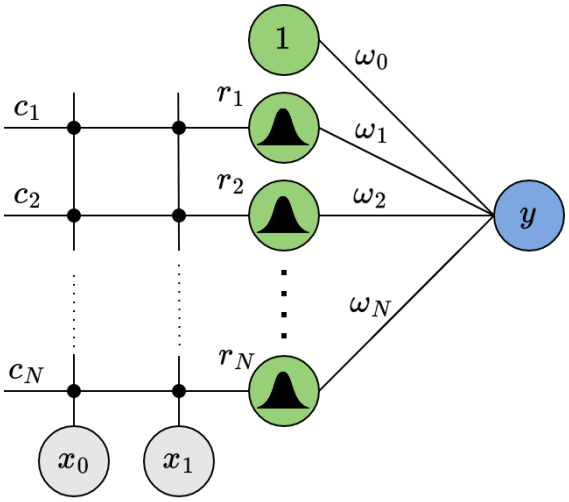


Figure 3.1: Exemplary structure for the proposed TinyRBF approach.

The hidden layer is the core of the network and contains a dynamic number of RBF neurons (or nodes) that use Gaussian activation functions. Each neuron in this layer has a center that identifies a specific point in the input space that the neuron "represents", and a radius (or standard deviation) that controls the area of influence of the neuron around its center. The Gaussian function produces higher activations for inputs close to the neuron's center and lower activations for more distant inputs, effectively capturing the relevance of each neuron for that input.

The output layer is a fully connected layer that combines the activation values from the hidden layer. In addition to the Gaussian RBF nodes, a bias node is also considered, whose activation values is constant and equal to 1. Each hidden neuron's activation is multiplied by a corresponding weight, and then summed to produce the final output.

3.3 Inference

Let i the number of inputs in the network, N the number of hidden neurons without considering the bias node, and O the number of outputs. When a new input $\boldsymbol{x}(t) \in$

$\mathbb{R}^I$ is presented to the network, it is passed to the RBF layer, where each neuron k has a defined center $\boldsymbol{c}_k \in \mathbb{R}^I$ and radius $r_k \in \mathbb{R}$, with $k = 1,\ldots,N$. For each node, the distance between the input and the neuron's center is calculated using Euclidean distance. This distance is then used in the Gaussian Radial Basis Function to compute the neuron's activation value $\theta_k(\boldsymbol{x}(t))$ as reported in Equation 3.7.

$$\theta_k(\boldsymbol{x}(t)) = \exp\left\{-\frac{\|\boldsymbol{x}-\boldsymbol{c}_k\|^2}{r_k^2}\right\}, \quad k = 1,\ldots,N \tag{3.7}$$

The activations from all RBF neurons together with bias activation $\theta_0(\boldsymbol{x}(t)) = 1$, are passed to the output layer. Each activation value is weighted by a corresponding parameter $\omega_{k,i}$, $k = 0,\ldots,N$, $i = 0,\ldots,O-1$, and a weighted sum of these activations is computed to produce the network's outputs $\boldsymbol{y}(t) \in \mathbb{R}^O$, as shown in Equation 3.8.

$$y_i(\boldsymbol{x}(t)) = \sum_{k=0}^{N} \omega_{k,i}\theta_k(\boldsymbol{x}(t)), \quad i = 0,\ldots,O-1 \tag{3.8}$$

In summary, the inference process of a TinyRBF network maps new inputs to outputs by evaluating how closely the input resembles the learned representations stored in each neuron, calculating activations based on distances, and linearly combining these activations to produce a prediction.

3.4 Learning Process

For learning to occur, it is essential to compute the model's prediction error, as this calculation determines the specific updates required in the model's parameters or network architecture to minimize the error effectively. For this reason, the learning process at instant t can be executed only when a target output $\hat{\boldsymbol{y}}(t)$ is available, and consists of the following steps.

1. The output of the network $\boldsymbol{y}(t)$ is calculated through inference: activation values of the hidden layer are computed using Equation 3.7, while Equation 3.8 is used to determine network outputs.

2. Model's prediction error at instant t, $\boldsymbol{e}(t)$, is computed with Equation 3.9.

$$\boldsymbol{e}(t) = \boldsymbol{y}(t) - \hat{\boldsymbol{y}}(t) \tag{3.9}$$

In case the hidden layer of the network is empty, i.e., $N = 0$, a single RBF unit is added in the model with characteristics defined by Equation 3.10, and the learning process then terminates.

$$\begin{cases} \boldsymbol{c}_1 = & \boldsymbol{x}(t) \\ r_1 = & r_{\text{init}} \\ \boldsymbol{\omega}_1 = & \boldsymbol{e}(t) \end{cases} \tag{3.10}$$

Here, r_{init} denotes the radius (or width) of the first hidden neuron allocated in the network and it is highly task dependent.

3. Assuming $N > 0$, the distance from the nearest hidden neuron, $d(t)$, is determined by Equation 3.11.

$$d(t) = \min_k \|\boldsymbol{x}(t) - \boldsymbol{c}_k\|, \quad k = 1, \ldots, N \tag{3.11}$$

4. A new RBF unit is introduced when the network encounters difficulty in accurately approximating the target function, or when the input differs substantially from previously encountered data. These requirements can be expressed with the condition in Equation 3.12. In this formula, ε represents the target prediction error and $\delta(t)$ is the dynamic distance threshold determined by the procedure described in Section 3.4.1.

$$\begin{cases} \|\boldsymbol{e}(t)\| > \varepsilon \\ d(t) > \delta(t) \end{cases} \tag{3.12}$$

When the allocation condition is verified, a new hidden neuron is inserted with the characteristics listed in Equation 3.13, where β is an overlap factor.

$$\begin{cases} \boldsymbol{c}_{N+1} = & \boldsymbol{x}(t) \\ r_{N+1} = & \beta\ d(t) \\ \boldsymbol{\omega}_{N+1} = & \boldsymbol{e}(t) \end{cases} \tag{3.13}$$

The new RBF node is centered on the current input, with its radius set to the distance from the nearest existing node modulated by β, and its weights adjusted to account for the remaining contribution needed to achieve the correct output (i.e., the prediction error).

5. In the case where the allocation of a new node in the network is not necessary, i.e., Equation 3.12 is not verified, the centers of the Gaussian RBF units and the weights of the linear output layer are updated. This is done through the use of the gradient descent method. Since the model has a depth of 2 layers, backward propagation of the error is not necessary; instead, closed-form parameter updating is possible, as reported in Equation 3.14.

$$\begin{cases} \boldsymbol{\omega}_0 \leftarrow & \boldsymbol{\omega}_0 + \eta\, \boldsymbol{e}(t) \\ \omega_{k,i} \leftarrow & \omega_{k,i} + \eta\, e_i(t)\, \theta_k(t) \\ \boldsymbol{c}_k \leftarrow & \boldsymbol{c}_k + \eta\, \frac{2\,\theta_k(t)}{r_k^2}\, (\boldsymbol{e}(t) \cdot \boldsymbol{\omega}_k)\, (\boldsymbol{x}(t) - \boldsymbol{c}_k) \end{cases} \tag{3.14}$$

The η hyper-parameter represents the learning rate.

6. The network pruning strategy is performed with the goal of removing neurons that no longer contribute sufficiently to the output computation. Formally, if the normalized activation of a hidden neuron k turns out to be less than a α threshold value for more than W consecutive training steps, that node is removed from the model.

$$\theta_k^{\text{norm}}(\boldsymbol{x}(t)) = \left| \frac{\theta_k(\boldsymbol{x}(t))}{\max_k \theta_k(\boldsymbol{x}(t))} \right|, \quad k = 1, \ldots, N \tag{3.15}$$

In detail, initially the normalization of RBF unit activations is performed as in Equation 3.15. Then, for each node it is checked whether its normalized activation is less than the α threshold, as reported in Equation 3.16.

$$\theta_k^{\text{norm}}(\boldsymbol{x}(t)) < \alpha, \quad k = 1, \ldots, N \tag{3.16}$$

If so, a counter associated with the hidden neuron is incremented; otherwise, it is set to zero. Finally, all nodes with a counter value greater than the pruning window W are removed from the network, effectively changing its topology.

7. Finally, the learning process returns the target output $\hat{\boldsymbol{y}}(t)$, as this is the best prediction value for instant t.

3.4.1 Adaptive Distance Threshold

According to several experimental results, it could be established that the distance threshold $\delta(t)$ is the most critical parameter governing the complexity of the evolving model. In detail, high threshold values could prevent the addition of new neurons and thus limit the predictive ability of the network, falling into the problem of *underfitting*. Conversely, too low threshold values would lead to continuous allocation of RBF nodes, bursting the complexity of the model and leading to the *overfitting* problem. Furthermore, it could be verified that using a fixed distance threshold value may be unsuitable throughout the evolution of the network over time.

For the various reasons described above, one of the most relevant feature of the proposed approach is a dynamic distance threshold that adjusts over time according to the distribution of distances between the input pattern and the nearest neuron. Based on experimental results, the formula in Equation 3.17 achieves an optimal balance between model complexity and the accuracy of function approximation.

$$\delta(t) = \mu_d(t) + \gamma\sqrt{\sigma_d(t)} \tag{3.17}$$

The hyper-parameter γ represents a multiplicative constant whose value needs to be determined through experimental tests so as not to excessively restrict network growth that would negatively impact the model's predictive performance.

Equation 3.17 reports how the threshold value $\delta(t)$ is computed at every learning step, where $\mu_d(t)$ and $\sigma_d(t)$ are the approximated mean and variance at instant t of the distance between the input pattern and the nearest neuron for the last M time steps. The update of these two values occurs at every inference using Equation 3.18, eliminating the need of storing the last M values in memory.

$$\begin{aligned} \mu_d(t) &= \mu_d(t-1) - \tfrac{\mu_d(t-1)}{M} + \tfrac{d(t)}{M} \\ \sigma_d(t) &= \sigma_d(t-1) - \tfrac{\sigma_d(t-1)}{M} + \tfrac{(d(t)-\mu_d(t))^2}{M} \end{aligned} \tag{3.18}$$

In other words, the values $\mu_d(t)$ and $\sigma_d(t)$ approximate the mean and variance of distance values using a moving average with a rolling window size of M. This approximation treats the oldest value in the window as equivalent to the current mean. From a complexity perspective, this approach enables the computation of two moving averages while requiring storage of only two parameters: $\mu_d(t)$ and $\sigma_d(t)$.

3.4.2 Hyper-Parameters

This section aims to list and analyze in detail the role of all the hyper-parameters that govern the learning process of a TinyRBF model, and thus its evolution over time.

Initial Neuron Radius r_{init} The hyperparameter r_{init} defines the radius (or width) of the first Gaussian RBF unit allocated in the network. This parameter determines the area of influence for the first neuron and plays an important role in modeling the initial behavior of the network. This parameter is only used when the initialization procedure is not employed, since the initialization process sets the centers and widths of the initial neurons explicitly. The value of r_{init} is highly dependent on the task, especially on the distribution of the input data. The choice of this radius affects the ability of the network to generalize and adapt to the data during subsequent learning. When the initial radius of the neuron is small, the area of influence of the first neuron is limited to a narrow region around its center. This can lead to a rapid degradation of the model's prediction performance, especially for input data that changes rapidly over time. Conversely, when the initial radius of the neuron is large, the first neuron has a large area of influence, helping the network generalize better in the early stages. However, it can result in a poor approximation of fine details in the input data, and it can also slow down the network's ability to adapt to dynamic or complex patterns over time.

Target prediction error ε The target prediction error specifies the acceptable level of error between the network output and the actual target value during learning. This parameter serves as a threshold to evaluate the accuracy of the network in approximating the desired function. This hyperparameter plays an important

role regarding the addition of neurons. Specifically, if the prediction error of the network on a new input exceeds this threshold, the addition of a new RBF neuron can be triggered. This new neuron is positioned to better represent the region of the input space where the error was high, thus reducing the overall error and improving the accuracy of the network, but at the same time increasing the complexity of the model. On the other hand, if the model error is less than the ε threshold, one can avoid increasing the number of hidden neurons and simply update the parameters already present.

Overlap factor β In Gaussian RBF networks, the overlap factor determines the degree of overlap between the Gaussian curve of neighboring units in the input space. Each RBF unit has a center and a radius (or standard deviation), which defines how far that unit's influence extends from its center. The overlap factor controls the balance between localization and generalization in the network. Specifically, when the overlap factor is high, neighboring RBF units cover larger regions and overlap more significantly. This greater overlap smooths transitions between units, helping the network to generalize better and provide more uniform approximations of the function. However, too much overlap can reduce the network's ability to capture fine details of the data. In contrast, with a lower overlap factor, each RBF unit covers a smaller region with less overlap with its neighbors. This results in a more localized response to the input data, allowing the network to capture detailed and specific patterns. However, lower overlap can lead to gaps between receptive fields, causing the network to be difficult to generalize and potentially requiring more units to adequately cover the input space.

Learning rate η In the gradient descent algorithm, the model updates its parameters in the direction that reduces the error, as determined by the gradient of the loss function with respect to each parameter. The learning rate determines the size of each of these steps. In particular, if the learning rate is too high, the optimization process may exceed the minimum or even diverge, leading to instability. If, on the other hand, the learning rate is too low, the steps will be very

small, causing slow convergence that may require a large number of iterations to reach an acceptable minimum or even stall at a local minimum. Choosing an appropriate learning rate balances these extremes, but may depend on other factors such as the frequency of references. Indeed, in the case of infrequent target outputs, a high learning rate may be preferred to promote model convergence, but at the same time more importance would be given to some possible measurement noise.

Distance threshold amplitude γ The computation of the adaptive distance threshold is based on the distribution of the distance between the input pattern and the closest neuron, using the approximate mean and variance of the latter, as reported in the equation 3.17. The hyperparameter γ adjusts the impact of the variance contribution in the formula. In detail, a high value of γ makes the threshold farther from the mean value and therefore disfavors the allocation of new neurons in the network. On the contrary, a low value of γ narrows the distance between the mean and the threshold, making the addition of RBF nodes easier.

Rolling window size M When calculating a moving average, the window size refers to the number of recent data points used to calculate the average at each step. The window size determines the number of values included in each calculation, which affects the smoothness and responsiveness of the moving average. For instance, with a large window, the moving average includes more data points, making it smoother and less sensitive to short-term fluctuations. On the other hand, a smaller window includes fewer data points, making the moving average more sensitive to recent changes and short-term variations. However, it can also make the average more volatile, as it can fluctuate with each new data point. Although in the case of the adaptive distance threshold, the two moving averages are approximated, the considerations still apply.

Activation threshold α The activation threshold for pruning is a hyperparameter that helps managing the complexity of the network by selectively removing neurons that do not contribute significantly to the performance of the model.

Its value represents the threshold that determines whether a particular contribution of an RBF node can be considered significant enough. Its value is to be understood as a relative measure: for example, assuming $\alpha = 0.01$, the activation of a neuron is not considered significant if it does not contribute at least 1% in the output calculation.

Pruning window size W The pruning window size refers to the period over which neuron activation values are monitored to decide if a node should be pruned. This hyper-parameter controls the time frame used to assess whether a neuron's contribution is consistently below the activation threshold α. In particular, a large window size makes the pruning process more conservative, reducing the likelihood of prematurely removing neurons that might still contribute under certain conditions. However, it can allow low-activation neurons to persist longer than necessary, leading to inefficiency and unnecessary complexity in the network. On the contrary, a small window size enables the network to be more responsive, quickly removing neurons that appear to have low relevance. Nevertheless, a low window size enables faster pruning but may risk premature removal of useful neurons.

3.5 Online Model Evolution

Figure 3.2 provides a conceptual overview of an on-device learning process for a TinyRBF network, emphasizing inference, learning, and model update steps over time.

Before deploying the model, the initialization process presented in Section 3.6 can be performed, which sets initial parameter values for the network, that are, centers and radii of initial Gaussian RBF units and weights of the output layer. This initialization can help stabilize early learning and improve network performance from the outset. However, there is also the possibility that the network is initially empty, that is, without hidden neurons. This means that there is no need to collect an initial dataset and to have prior knowledge in the model.

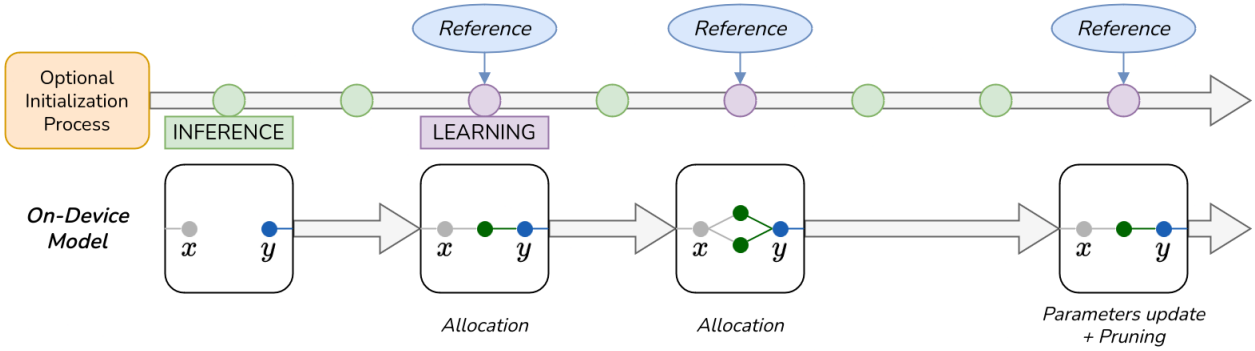


Figure 3.2: Online inference and learning process. In the upper part, time is marked by sensor sampling where inference is performed, if a reference is not available, or the model is updated otherwise. Below, an example of network evolution is shown.

The diagram flows horizontally, illustrating a sequence of inference and learning steps as the model processes data over time. Each inference step (shown in green) represents a point where the model is used to predict an output $\boldsymbol{y}(t)$ for a given input $\boldsymbol{x}(t)$. While the learning steps (shown in purple) are triggered when the model receives a reference value, i.e., the correct output data that is used to adapt and improve the network's parameters or structure. During the learning phase, neuron allocation may occur. Allocation refers to adding new Gaussian RBF neurons to the hidden layer when the current model structure is not sufficient to accurately approximate the desired output.

The bottom part of the image shows the model evolving over time as neurons are added based on new data. In the first learning step, only one allocation can occur. As new data points are introduced, more neurons (green dots) are allocated, each centered on a new reference point. In the final learning step shown on the right, the network undergoes parameter updates and pruning. Parameter updates adjust the centers, radii, and weights of existing neurons to minimize prediction error and adapt to changes in the input data, without changing the network topology. Pruning removes neurons that contribute minimally to the output, exhibiting low activation value. This

phase keeps the model compact, improving computational efficiency without sacrificing accuracy.

In a real case, the frequency of the learning steps is estimated much lower than reported in the figure, since the operation of obtaining the external reference value is considered expensive and difficult to perform.

3.6 Initialization

This section describes an initialization process that can be optionally performed for the purpose of defining a network structure starting from a dataset. In contrast to the online phase in which inference and learning process is performed, this procedure has to be considered "offline" in the sense that it does not take deployment requirements into account. This method not only performs batch learning, but has a complexity that makes it unsuitable for a device with severe resource constraints.

Given an initial number of hidden neurons equal to N, the initialization process includes three steps that aim to determine the values of the centers and radii of the Gaussian RBF nodes and the weights in the linear output layer, similarly to what is described in [57].

First, the centers of RBF c_k nodes are determined from the centroids of N clusters computed by applying the K-means algorithm on all input data. Then, the radius of each hidden node r_k is determined by the distance from the nearest center, as reported in Equation 3.19.

$$r_k = \|c_k - c_{m'}\|, \quad m' = \underset{1 \leq m \leq N, m \neq k}{\arg\min} \|c_k - c_m\| \tag{3.19}$$

In addition to the N nodes thus generated, a bias node, whose activation value is 1, is also considered.

$$\boldsymbol{W} = (\boldsymbol{\Omega}^T \boldsymbol{\Omega})^{-1} \boldsymbol{\Omega}^T \boldsymbol{Y} \tag{3.20}$$

Finally, the weights of the output layer are computed by the least squares method, as shown in Equation 3.20, where $\boldsymbol{W} = [\omega_{k,i}]$ is the matrix of weights, $\boldsymbol{Y}$ is a matrix containing in each row the output data of the training set, and $\boldsymbol{\Omega}$ is a matrix whose rows are the activations of the input data computed as in Equation 3.7.

3.7 Complexity Analysis

Let I represent the number of input features, N the number of hidden neurons, and O the number of outputs of the network.

In terms of network parameters, each neuron in the hidden layer has a center — an I-dimensional vector — and a radius. The bias node is an exception, as it has no parameters due to its constant output. The number of activations in the final layer is $N+1$, corresponding to the hidden layer neurons plus the bias, with O as the total number of outputs. Moreover, the pruning strategy requires an additional counter for each hidden neuron. In this way, the overall number of parameters for the network can be computed using Equation 3.21.

$$(I+1)\times N+(N+1)\times O+N \tag{3.21}$$

Using a 32-bit floating point representation, the memory footprint in bytes is calculated by multiplying the expression in Equation 3.21 by 4 bytes. In practice, the bit size of the pruning counter can be reduced based on its maximum possible value, W. For example, if $W < 256$, an 8-bit unsigned integer may be adequate for this purpose.

In terms of time complexity, inference involves first calculating activations using Equation 3.7, followed by computing outputs through the fully connected layer as shown in Equation 3.8. For each hidden node, calculating the distance between the input and the center of a unit requires I subtractions and I multiplications. Next, the radius is squared, inverted, negated, and multiplied by the result from the previous step. The exponential function is then applied; however, its complexity C_{exp} remains undefined, as implementation details and platform-specific optimizations can affect the number of operations required. For Equation 3.8, each output requires N multiplications and $N+1$ additions. Overall, the time complexity of the model during inference, expressed in Floating Point Operations (FLOPs), is given by Equation 3.22.

$$C_{\text{inf}} = N\times(C_{\text{exp}}+2I+4)+O\times(1+2N) \tag{3.22}$$

The time complexity analysis of the learning algorithm, as outlined in Section 3.4, focuses on its critical path — the sequence of operations that requires the most time or

Inference	*Refer to Equation 3.22*
Prediction error computation	O
Empty network check	1
Nearest neuron distance computation	$N(3I+1)$
Allocation condition	$2O+2$
Parameters update	$N(4O+2I+5)+2O$
Pruning strategy	$5N$

Table 3.1: Details of critical path time complexity of learning process steps.

resources to complete. Activities included in this path are listed in Table 3.1, together with their time complexity. The overall time complexity of the learning process is provided in Equation 3.23 and expressed in FLOPs.

$$C_{\text{learn}} = C_{\text{inf}} + N \times (4O + 5I + 11) + 5O + 3 \tag{3.23}$$

Specifically, this includes inference computation, prediction error and nearest neuron distance computation, evaluation of allocation conditions, and finally, the application of network parameter updates and pruning strategies.

Chapter 4

Experimental Evaluation

This chapter provides a comprehensive presentation and analysis of all experiments conducted to validate and quantify the performance of the solutions proposed in this Thesis. Each application is examined in detail, starting with an overview of the sensors used, including their specifications and applied stresses, followed by a description of the data collection conditions. The dataset characteristics are also outlined, covering the structure, sampling frequency, number of measurements and any performed data pre-processing step. Next, the evaluation metrics employed and validation techniques are defined to establish a clear basis for performance comparison. Each section conclude with an analysis of the experimental results, discussing the accuracy, memory efficiency, and computational demands of the proposed specific solution.

The predictive performance of the proposed models was assessed using multiple metrics, selected for their relevance to the characteristics of the dataset and the objectives of the analysis [58]. The *Mean Absolute Error* (MAE) measures the average absolute difference between predicted and actual values, providing a direct indication of model accuracy. Alongside this, the *Root Mean Squared Error* (RMSE) emphasizes larger errors by computing the square root of the average squared differences between predictions and true values. Additionally, the *Symmetric Mean Absolute Percentage Error* (SMAPE) was considered, which quantifies the relative absolute difference

between predicted and actual values in percentage terms, accommodating both overestimation and underestimation. The definition of these three metrics is reported in Equations 4.1, 4.2 and 4.3, respectively, where n is the number of samples, y_t represents the actual value and $\hat{y}_t$ represents the predicted value.

$$\text{MAE} = \frac{1}{n}\sum_{t=0}^{n}|y_t - \hat{y}_t| \tag{4.1}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=0}^{n}(y_t - \hat{y}_t)^2} \tag{4.2}$$

$$\text{SMAPE} = \frac{1}{n}\sum_{t=0}^{n}\frac{|y_t - \hat{y}_t|}{(|y_t| + |\hat{y}_t|)/2} \cdot 100\% \tag{4.3}$$

The chapter is organized as follows. Section 4.1 reports the experimental results of the implementation and deployment of the six different approaches used to solve the model describing the parameters of the Bioristor sensor (*Use Case 1*). The results of the application of the TinyRBF algorithm to the *Use Case 2* during all the life phases of an atmospheric pressure sensor are analyzed in Section 4.2. While Section 4.3 describes the field experimentation for the self-calibration of several environmental sensors of pressure, temperature and humidity (*Use Case 3*). Finally, Section 4.4 enriches the results with two applications of the TinyRBF approach outside the context of precision agriculture, namely a glucose sensor and Inertial Measurement Units (IMUs).

4.1 Bioristor Data Analysis

This section reports experimental data from the application of the five different methods for solving the mathematical model of an OECT sensor described in Chapter 2. Experimental results in terms of accuracy, memory usage, execution time, and energy efficiency reported in this section, were also published in the paper entitled *"Energy-efficient OECT Sensor Data Analysis on Constrained Edge Devices"* [59].

Section 4.1.1 describes the hardware characteristics of the development boards targeted for deployment. Section 4.1.2 introduces the dataset employed for experiments, reporting details about samples collection and data preprocessing. Finally, Section 4.1.3 analyzes the experimental results obtained from the point of view of memory footprint, computational complexity, precision and energy consumption.

4.1.1 Experimental Setup

The experiments were conducted using two distinct development boards: the NUCLEO-L476RG and the NUCLEO-F767ZI, both from STMicroelectronics. These boards provide an ultra-low-power platform and a high-performance platform, respectively, for assessing memory usage, execution time, and overall energy consumption.

The NUCLEO-L476RG board is equipped with the STM32L476RG chip, an ultra-low-power microcontroller featuring the high-performance ARM Cortex-M4 32-bit RISC core, which operates at frequencies of up to 80 MHz. It includes a single-precision floating-point unit (FPU), 128 KB of RAM, and 1 MB of Flash memory.

In contrast, the NUCLEO-F767ZI board is built around the STM32F767ZI chip, which is based on the high-performance ARM Cortex-M7 32-bit RISC core and operates at frequencies of up to 216 MHz. This microcontroller features both single-precision and double-precision floating-point units, along with 786 KB of RAM and 2 MB of Flash memory.

4.1.2 Dataset

Data for algorithm validation was collected by installing Bioristor sensors in 28 tomato plants and monitoring their response continuously over five days. To introduce variability into the dataset, two irrigation treatments were used: fourteen plants were regularly watered, while the other fourteen were drought-stressed by withholding irrigation. It is important to clarify that the goal of these two treatments was not to create a binary-labeled dataset (watered vs. drought-stressed). Instead, the objective was to include greater variability, allowing the algorithm to effectively capture a wide range of physiological responses.

Due to the hand-crafted nature of these sensors, R_{dry} values varied between 20 Ω and 70 Ω, though the sensor response and its dependence on concentration (C) remained consistent. Each sensor's R_{dry} was recorded and used to calibrate the model. The gate duty cycle was set to 50% with a total cycle period of 30 minutes. After each gate cycle, I_{ds}^{on}, I_{ds}^{off}, and I_{gs}^{on} were extracted from the raw data and recorded with timestamps in a CSV file.

From this initial data, comprehensive preprocessing was conducted to remove outliers and invalid values, resulting in a dataset of 18,365 samples. The extracted current values and R_{dry} were combined with the model variables calculated using Newton's method, yielding the dataset for training and testing the multilayer perceptron model. Newton's method was chosen for its ability to achieve the lowest average error.

Standardization was applied to each feature and target output in the dataset, centering values around the mean with a standard deviation of one. These parameters were determined from the distributions of neural network inputs and target outputs, as shown in Figure 4.1 and Figure 4.2.

4.1.3 Results

The algorithms outlined in Section 2.3 were evaluated based on memory footprint, execution time, power consumption, and solution accuracy. It is important to emphasize that the primary objective of this experimentation was the specific characteriza-

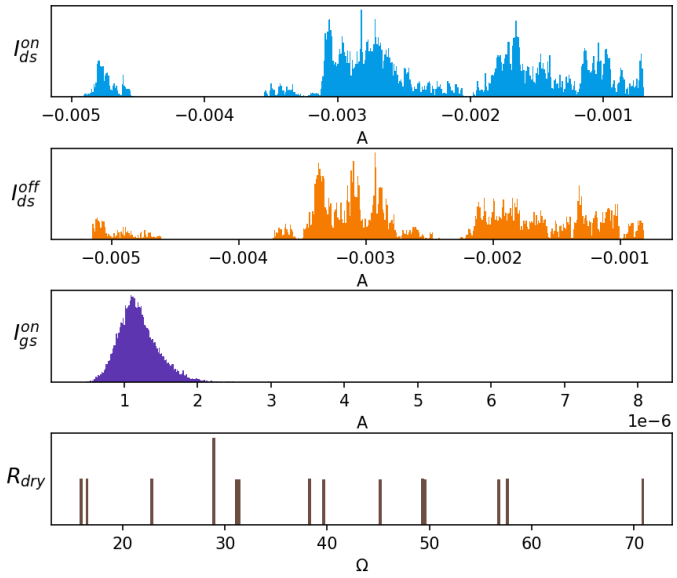


Figure 4.1: Distribution of the neural network inputs.

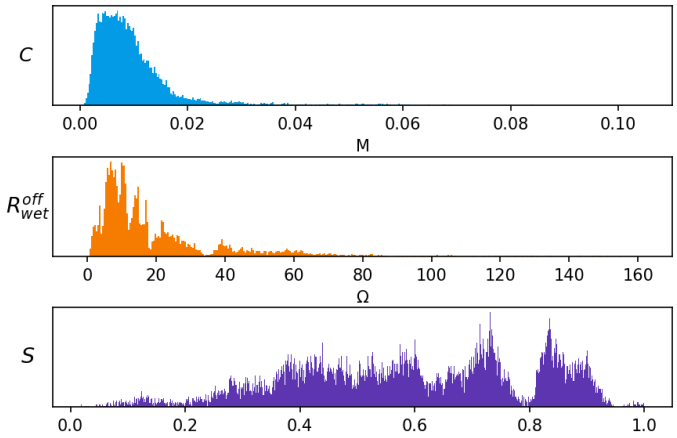


Figure 4.2: Distribution of the neural network target outputs.

tion of the Bioristor, rather than a comparison of its performance with other sensors. Furthermore, it should be highlighted that the Bioristor had never been characterized in this way before, making this study an innovative contribution to its understanding and application. For reliable statistics in terms of MCU execution, each program runs a single algorithm with fixed input currents and includes only the essential code to initialize the development board.

Memory usage in RAM and Flash was analyzed with the *cargo-binutils* tool, which reports the compiled binary size in Berkeley format. Specifically, it provides byte sizes for three sections: *text*, containing the code and constants in Flash memory; *data*, referring to initialized variables counted in both RAM and Flash; and *bss*, for uninitialized data in RAM. As no heap or dynamic memory was used, the total RAM utilization is calculated by summing the *data* and *bss* sections, while total Flash usage is the sum of the *text* and *data* sizes.

Execution time and CPU cycles were measured using a profiler based on the *SysTick* 24-bit counter, commonly available in ARM Cortex-M devices. To address potential overflows during lengthy tasks, the profiler enhances *SysTick* resolution by incorporating a 64-bit counter that increments with each overflow, triggered every 2^{24} clock cycles.

Power consumption measurements for the development boards were conducted using the Power Profiler Kit II by Nordic Semiconductor. This tool offers a measurement range from 200 nA to 1 A, with a resolution that varies from 100 nA to 1 mA based on the measurement interval, and supports a sampling frequency of up to 10^5 samples per second. It functions as both an ammeter and a power source, supplying voltages from 0.8 to 5 V.

Each solution's error was determined by calculating the deviation from zero in Equation 2.5, substituting the output variables produced by the algorithm.

The parameters used for the experiments on each evaluated algorithm are shown in Table 4.1. For the MLP, two specific topologies were selected following an extensive hyperparameter tuning phase: one with a single hidden layer of 16 neurons, and another with two hidden layers comprising 64 and 32 nodes, respectively. The ReLU [60] activation function was chosen due to its effectiveness in reducing loss

Brute Force Approach	
Concentration range	$[10^{-4}, 10^{-1}]$
Number of steps	100000
Adaptive Algorithm	
Concentration range	$[10^{-4}, 10^{-1}]$
Number of steps	1000
Maximum number of iterations	10
Number of minima	10
Reduction factor	0.2
Error tolerance	10^{-15}
Gradient Descent Algorithm	
Initial value of concentration	10^{-2}
Initial value of learning rate	0.1
Maximum number of iterations	10
Gradient tolerance	10^{-9}
Error tolerance	10^{-15}
Newton's Method	
Initial value of concentration	10^{-2}
Maximum number of iterations	10
Gradient tolerance	10^{-9}
Error tolerance	10^{-15}

Table 4.1: Parameters of the evaluated algorithms

and its straightforward implementation on a microcontroller. Both MLP models were trained for up to 200 epochs, using Stochastic Gradient Descent (SGD) with a learning rate of 0.05 as the optimizer and Root Mean Square Error (RMSE) as the loss function. The dataset was randomly divided into three subsets: 70% allocated for training, 20% for testing, and 10% for validation. This split enabled continuous monitoring of model performance during training and supported early stopping if necessary. Although the collected data represented a time series, the temporal dimension was not considered in the analysis; each sample was treated independently without correlation to its timestamp.

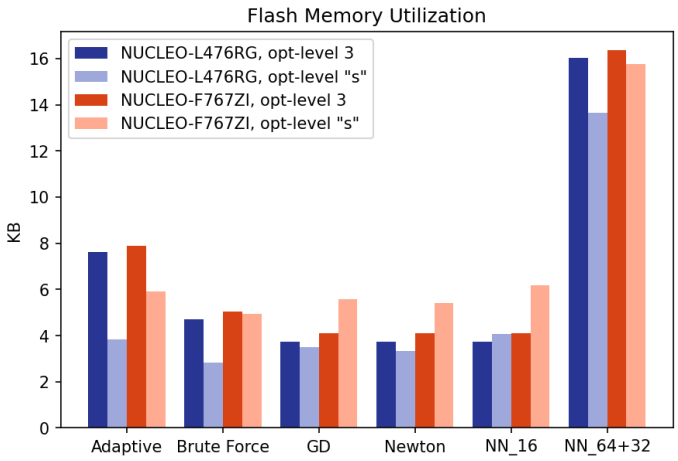


Figure 4.3: Flash memory utilization calculated as the sum of *text* and *data* sector sizes. GD stands for gradient descent, while NN_16 and NN_64+32 refer the two topologies of neural network.

Figure 4.3 illustrates the Flash memory consumption for each evaluated algorithm. The memory requirements for gradient descent, Newton's method, and the single-hidden-layer neural network are approximately 4 kilobytes each. In contrast, the brute-force and adaptive algorithms necessitate additional memory of 1 and 4 kilobytes, respectively. The neural network with two hidden layers requires over 14 kilobytes of Flash memory, primarily to accommodate its 2499 weights and biases. The "s" optimization option typically resulted in greater reductions in flash mem-

ory usage on the NUCLEO-L476RG board. For instance, it made the Brute Force algorithm less memory-intensive compared to Gradient Descent. The RAM usage remained consistent across all experiments, measuring 1,096 bytes for the NUCLEO-L476RG board and 1088 bytes for the NUCLEO-F767ZI board.

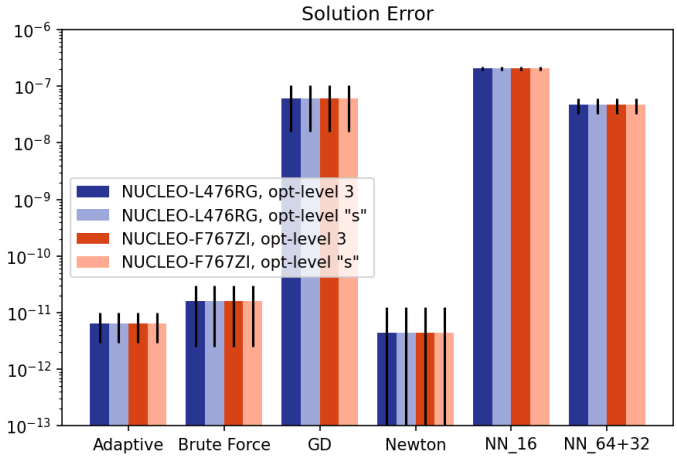


Figure 4.4: Mean solution error and its standard deviation after outliers removal.

The errors of the computed solutions were found to be minimal (less than 10^{-6}) across all evaluated algorithms, as shown in the plot in Figure 4.4. Specifically, gradient descent and the multilayer perceptron (MLP) exhibit error values around 10^{-7}, whereas the adaptive, brute force, and Newton's method demonstrate error values near 10^{-11}. Additionally, it is important to note that variations in chips and build configurations do not influence the accuracy of the solutions.

Given a tolerable limit for real-time processing of one millisecond, only the adaptive and brute-force approaches exceeded that threshold. Significant differences in execution times were observed among the various algorithms: the adaptive and brute force methods required more than 100 milliseconds to generate an output, while the other approaches completed the calculations in less than 1 millisecond, as illustrated in Figure 4.5. Notably, the neural network with a single hidden layer containing 16 neurons achieved an execution time of less than 10 microseconds on both development boards. The plot illustrating CPU cycles required for performing inference in

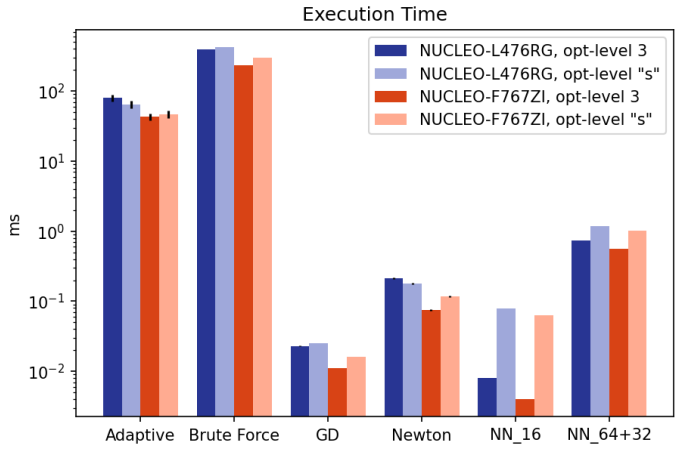


Figure 4.5: Mean execution time and its standard deviation.

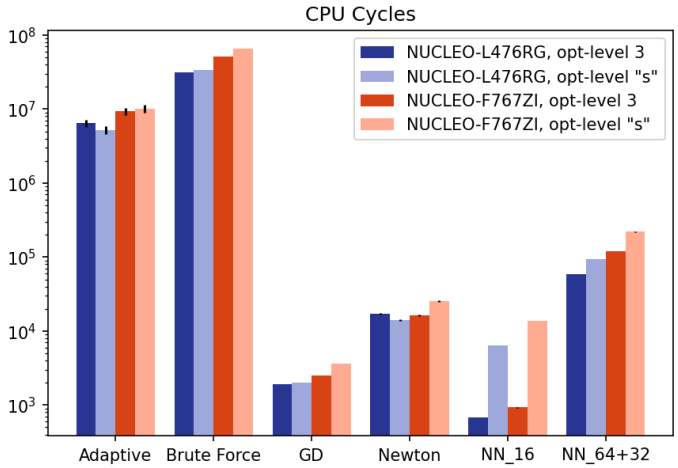


Figure 4.6: Mean number of CPU cycles required for execution and its standard deviation.

Figure 4.6 shows patterns similar to those observed in the execution time analysis of the evaluated algorithms. Notably, execution on the NUCLEO-F767ZI development board generally requires more cycles but achieves shorter execution times. This discrepancy can be attributed to the higher core frequency of 216 MHz on the NUCLEO-F767ZI, compared to 80 MHz on the NUCLEO-L476RG.

Concerning the optimization levels of the Rust compiler, it is generally observed that the “s" option results in smaller binaries while negatively impacting performance. Nevertheless, there are cases where this trend does not hold true, indicating the necessity of experimenting with various optimization levels to achieve the optimal balance for each application.

Board	NUCLEO-L476RG				NUCLEO-F767ZI			
Optimization level	3		“s"		3		“s"	
Algorithm	**Time [ms]**	**Energy [mJ]**	**Time [ms]**	**Energy [mJ]**	**Time [ms]**	**Energy [mJ]**	**Time [ms]**	**Energy [mJ]**
Adaptive Algorithm	80.01189	4.56868	64.74322	3.69684	43.14171	19.62516	47.03700	21.39713
Brute Force Approach	391.12569	22.33328	421.59141	24.07287	236.09061	107.39762	301.49528	137.15020
Gradient Descent	0.02301	0.00131	0.02500	0.00143	0.01100	0.005	0.01600	0.00728
Newton’s Method	0.21309	0.01217	0.17720	0.01012	0.07470	0.03398	0.11649	0.05299
Neural Network (1 hidden layer)	0.008	0.00046	0.07900	0.00451	0.00400	0.00182	0.06298	0.02865
Neural Network (2 hidden layers)	0.734	0.04191	1.17200	0.06692	0.55735	0.25354	1.01466	0.46157

Table 4.2: Power Consumption using a Power Source of 5 Volt

The measured current consumption remained consistent across the analyzed algorithms but varied significantly between the two development boards. It was observed that when powered by a 5 Volt source, the NUCLEO-L476RG has an average current draw of 11.42 mA, whereas the NUCLEO-F767ZI exhibits a mean current consumption of 90.98 mA. This results in instantaneous power consumption of 57.1 mW for the low-power board and 454.9 mW for the high-performance board.

Table 4.2 presents the mean execution time in milliseconds and the average energy consumed in mJ for each algorithm, board, and build configuration. The results indicate that, on average, execution on the NUCLEO-L476RG takes approximately

1.67 times longer than on the NUCLEO-F767ZI. Conversely, the NUCLEO-L476RG consumes only about 21% of the energy compared to the NUCLEO-F767ZI.

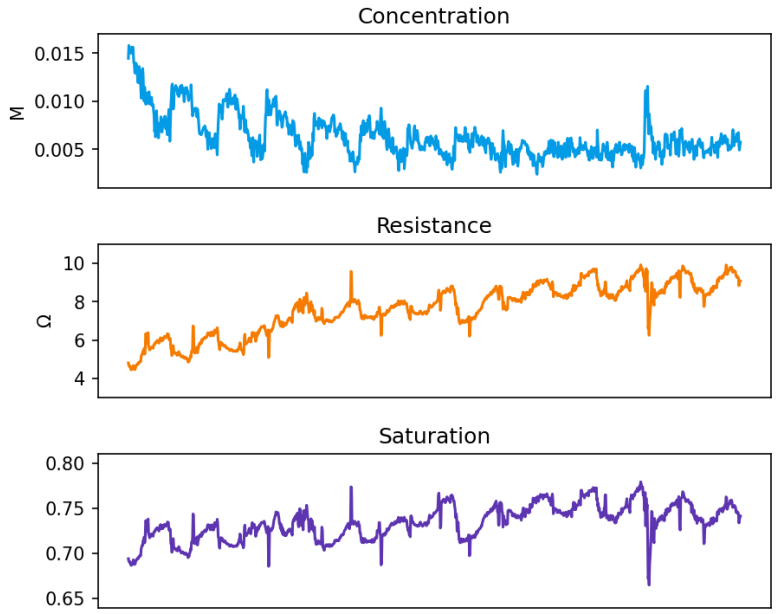


Figure 4.7: An example of the output values obtained by means of the Newton's method. Apart from local oscillations due to measurement noise, the trend of the values due to the day and night cycle is clearly visible.

4.2 Continuous Calibration of Pressure Sensors

This section reports and analyzes the experimental results related to *Use Case 2*, as described in Section 1.2.2. Some of the reported results, specifically those related to Dataset 2 and Dataset 4, have also been published in the papers *"In-Sensor Learning for Pressure Self-Calibration"* [61] and *"Learning Pressure Sensor Drifts from Zero Deployability Budge"* [62], respectively.

The sensors used in the laboratory experiments and the datasets employed in the three life phases of the latter ones are described in detail in Sections 4.2.1 and 4.2.2 respectively. Subsequently, starting from sensor manufacturing to end-user usage, the results regarding each step are analyzed in Section 4.2.3, 4.2.4 and 4.2.5.

4.2.1 Sensors

This section provides an overview of the LPS22HH and the newer and more precise LPS22DF atmospheric pressure sensors. Their key features and capabilities are described in detail to establish their relevance and suitability for the use cases addressed in this work.

LPS22HH

The LPS22HH[1] is a low-power digital barometric pressure sensor by STMicroelectronics designed for a variety of applications, including mobile devices, wearables, and IoT devices. It offers a digital output via I2C, MIPI I3CSM or SPI interfaces, facilitating easy integration with microcontrollers.

This sensor measures atmospheric pressure within a range of 260 to 1260 hPa, while also providing temperature measurements from -40°C to +85°C. LPS22HH achieves an absolute pressure accuracy of $\pm$0.5 hPa with a sensor noise of 0.65 Pa, whereas the absolute accuracy for temperature is $\pm$1.5 °C. With a pressure resolution of 0.01 hPa and a temperature resolution of 0.1 °C, the LPS22HH ensures precise data collection.

[1] `https://www.st.com/en/mems-and-sensors/lps22hh.html`

Its low power consumption of up to 4 µA makes it suitable for battery-powered devices and it has several power modes for greater energy efficiency. The conjunction of a built-in FIFO and a data output rate of 1 Hz to 200 Hz facilitates efficient data handling and reduces the processing load on the host microcontroller. Operating with a voltage range of 1.7 to 3.6 V, the LPS22HH comes in a compact LGA package, making it ideal for applications such as weather stations, altitude measurement, indoor navigation and smart home devices.

LPS22DF

The LPS22DF[2] is a low-power and high-precision digital barometric pressure sensor developed by STMicroelectronics. This sensor provides lower pressure noise, higher absolute precision and lower power consumption than its predecessor LPS22HH. Furthermore, this sensor features a digital output that can communicate via I2C, MIPI I3CSM, or SPI interfaces, enabling seamless integration with various microcontrollers.

The LPS22DF operates within a pressure measurement range of 260 to 1260 hPa and provides an absolute accuracy of $\pm$0.2 hPa and a sensor noise of 0.34 hPA, making it suitable for precise atmospheric pressure measurements. It also includes temperature sensing capabilities, with a measurement range from -40°C to +85°C and an accuracy of $\pm$1.5 °C.

Designed for low-power consumption, the LPS22DF is ideal for battery-operated applications, featuring a current consumption down to 1.7 µA and multiple low-power modes to enhance energy efficiency.

The combination of an embedded FIFO and an output data rate from 1 Hz to 200 Hz facilitates efficient data management and reduces the processing burden on the host microcontroller. Operating from a voltage range of 1.7V to 3.6V, the LPS22DF comes in a compact LGA package, making it well-suited for diverse applications including weather monitoring, altitude detection, indoor navigation, and smart home systems.

[2] https://www.st.com/en/mems-and-sensors/lps22df.html

4.2.2 Datasets

This section provides a detailed description of the four datasets used in the experiment. Each dataset differs in key characteristics, including the used type of sensor, the applied stresses, and the sampling duration, highlighting the comprehensive scope of the experiment.

The datasets were provided by the MEMS and Sensors Validation Laboratory at STMicroelectronics. Due to their proprietary nature, they are not publicly accessible.

Dataset 1 - LPS22HH in Post-Soldering Conditions

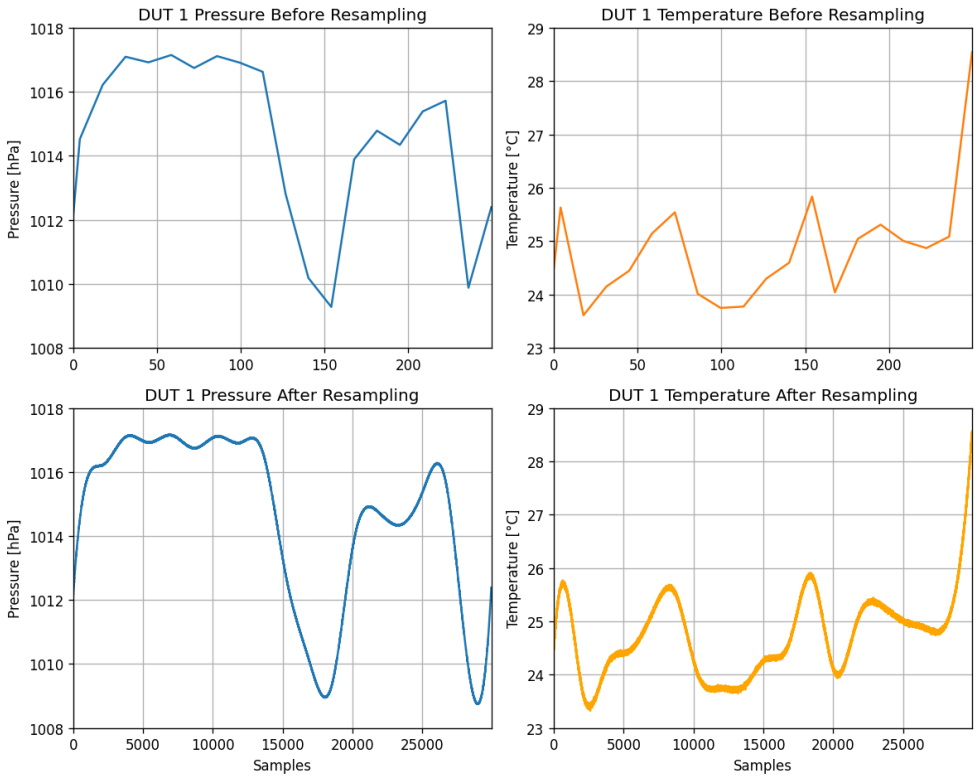


Figure 4.8: Comparison of pressure and temperature values for Device Under Test 1 in Dataset 1 before and after 30-second resampling.

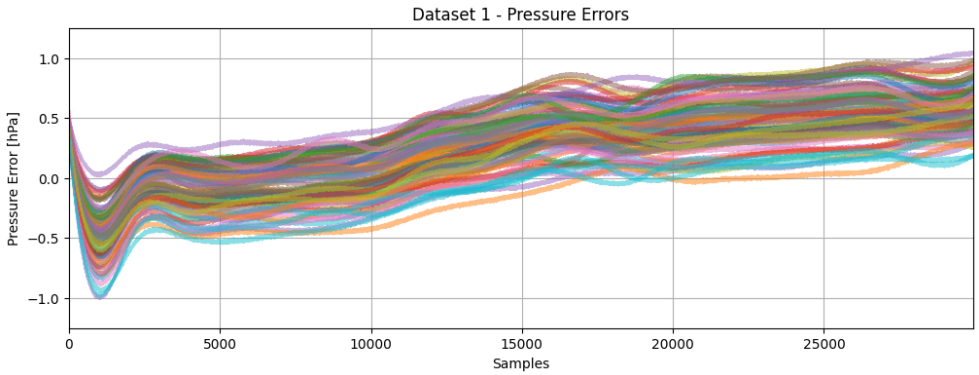


Figure 4.9: Trend of pressure errors for all sensors in Dataset 1.

The original version of Dataset 1 consists of pressure and temperature measurements collected by 80 LPS22HH sensors. Additionally, the dataset includes measurements from a high-precision, externally sourced reference sensor that was not part of the experiment. These measurements were recorded with a non-uniform collection frequency, averaging one sample every 13 hours. To align the sampling frequency with the other datasets, which provide data at regular intervals of one measurement every 30 seconds, an interpolation process using cubic splines was performed. This method ensured a uniform and consistent temporal distribution of the data points. To preserve the realism of the dataset and avoid generating artificially noise-free data, Gaussian noise was added to both the pressure and temperature measurements. The standard deviation for this noise was determined based on the specifications provided in the LPS22HH technical manual, ensuring that the simulated noise is as close as possible to the real behavior of the sensor. This preprocessing step not only standardized the dataset, but also maintained the integrity of the sensor's intrinsic variability, making it suitable for comparison with the other datasets used in the experiment. Figure 4.8 shows an example of comparison between the original pressure and temperature values obtained as a result of the oversampling operation. After the resampling phase, the dataset contains 29941 measurements, with a total duration of 11 days and 9.5 hours. The dataset variability and measurement error trends of all 80 sensors are shown in Figure 4.9.

Just before data collection, all sensors were subjected to a welding process that exposed them to severe thermal stress. However, during data collection, these sensors operated under normal operating conditions.

Dataset 2 - LPS22HH in Normal Conditions

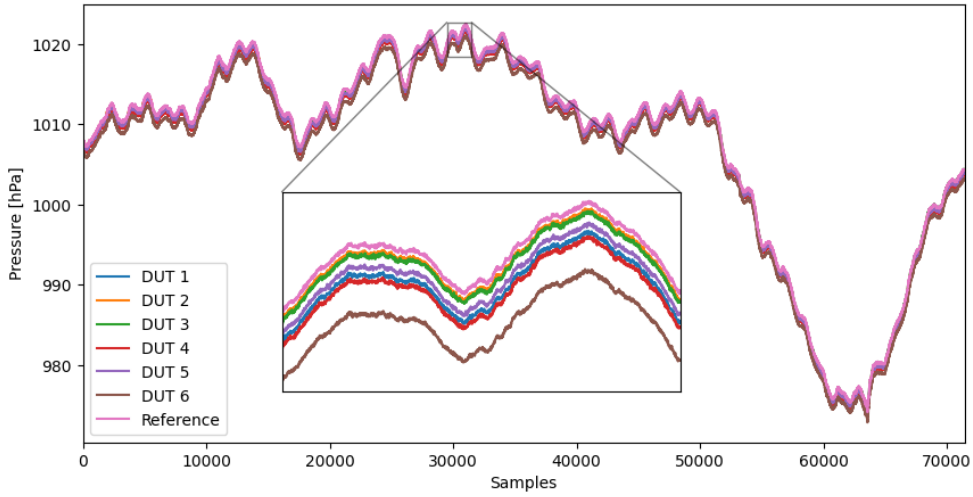


Figure 4.10: Overview of pressure measurements for Dataset 2.

Dataset 2 includes pressure and temperature readings from six LPS22HH sensors along with a reference pressure value. Each Devices Under Test (DUT) provided 71,428 samples recorded at a sampling rate of 30 seconds, covering nearly 25 days of data.

The dataset represents measurements collected from sensors operating exclusively under normal conditions, without any stress applied at any stage of the experiment. Unlike Dataset 1, these sensors were not subjected to thermal, mechanical, or environmental stresses either before, during, or after the data collection process. The data collection was performed in a controlled environment designed to ensure stable and ideal conditions for the sensors. This included maintaining stable ambient temperature, humidity, and pressure, as specified within the operational range of

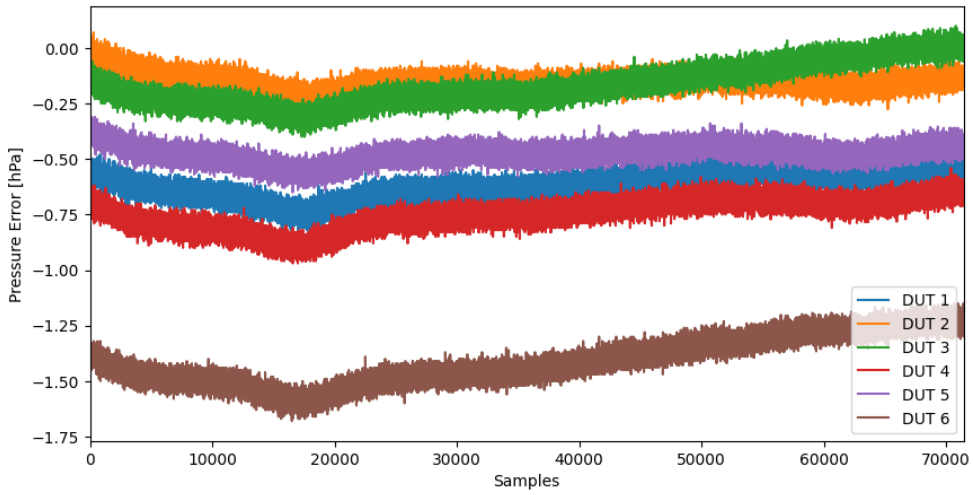


Figure 4.11: Pressure errors of Dataset 2 with respect to the reference value.

the sensors. Specifically, temperature was always between 18 and 23 °C, while the pressure varied in the range of 972-1023 hPa.

Figure 4.10 provides an overview of the pressure fluctuations recorded by the DUTs and the reference sensor. Additionally, Figure 4.11 illustrates the pressure error trend for each sensor relative to the reference value over the experiment duration. Baseline error metrics, computed using MAE, RMSE, and SMAPE, are summarized in the second row of Table 4.3.

Dataset 3 - LPS22HH in Extreme Stress Conditions

Dataset 3 was created by collecting data from 92 LPS22HH pressure sensors exposed to simulated extreme operating conditions in a controlled laboratory environment. Specifically, the sensors were subjected to cyclical changes in temperature, ranging from 5°C to 88°C, and pressure, ranging from 535 hPa to 1265 hPa, as shown in Figure 4.12. The primary goal of this dataset is to evaluate sensor performance and reliability under conditions that mimic real-world extremes, such as those encountered in harsh environments or during rapid environmental changes. This enables the

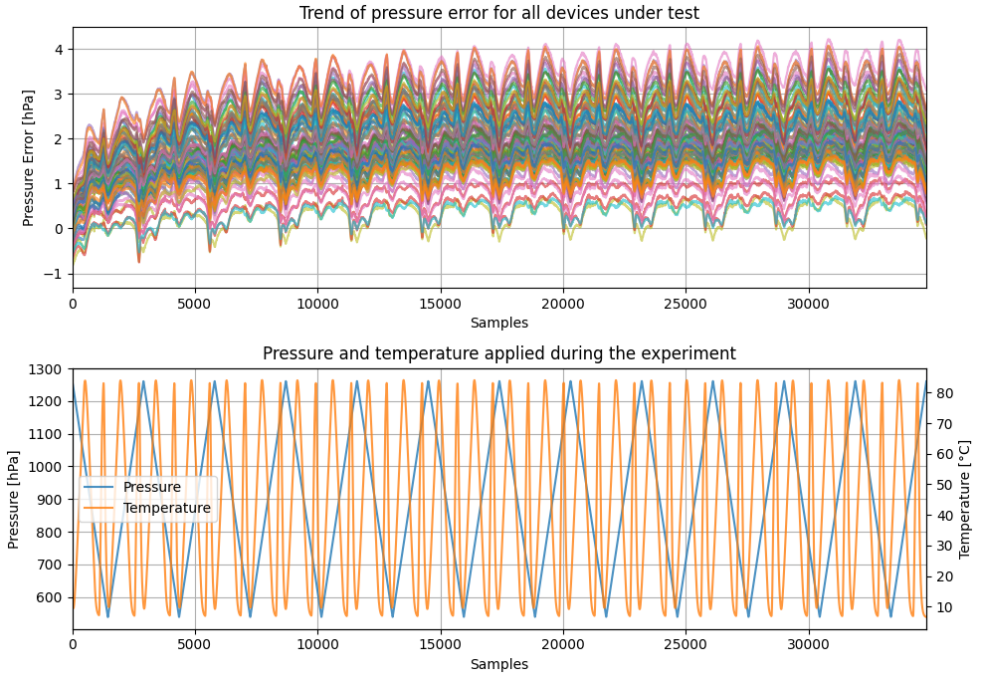


Figure 4.12: Above, the trend of pressure error for all 92 DUTS in Dataset 3, while below, the pressure and temperature applied during the entire duration of the experiment.

identification and characterization of nonlinear sensor behaviors, including hysteresis and drift, under varying stresses.

Temperature and pressure values were collected from the DUTs a total of 34800 times, with a sampling frequency of one measurement every 30 seconds. This corresponds to a total duration of approximately 12 days and 2 hours. In addition to the data collected from the LPS22HH sensors, the dataset includes reference values, often referred to as a "golden reference", obtained from a high-quality, expensive sensor. This reference sensor provided precise and reliable measurements of temperature and pressure, serving as a benchmark for evaluating the performance of the LPS22HH sensors under extreme conditions. Base errors of the dataset in terms of MAE, RMSE and SMAPE are reported in Table 4.3.

Figure 4.12 illustrates the pressure error trends for each sensor included in Dataset 3. The data reveals a chaotic pattern, mainly due to the sudden changes in environmental conditions to which the devices were subjected. This behavior contrasts with the more stable variations seen in Dataset 2, as shown in Figure 4.10. The absolute error values in Dataset 3 also cover a wider range, with noticeable differences between individual sensors. This highlights the importance of having a solution that can adapt to the specific drift behavior of each sensor, ensuring accurate and reliable measurements even under changing conditions.

Dataset 4 - LPS22DF in Different Thermal Stresses

Dataset 4 includes data from 29 state-of-the-art LPS22DF sensors exposed to five distinct thermal stress conditions. Pressure and temperature readings from the Devices Under Test (DUTs) were sampled 22919 times at 30-second intervals, resulting in a nearly 8-day dataset. Based on the specific thermal stress applied, the sensors were organized into six groups:

- Group 1 (DUTs 1-5): Exposed to -20 °C for 2.5 hours.
- Group 2 (DUTs 6-10): Exposed to 100 °C for 2 hours.
- Group 3 (DUTs 11-15): Exposed to 100 °C for 1 hour.
- Group 4 (DUTs 16-20): Exposed to -20 °C for 1 hour.
- Group 5 (DUTs 20-24): Exposed to 60 °C for 90 minutes.

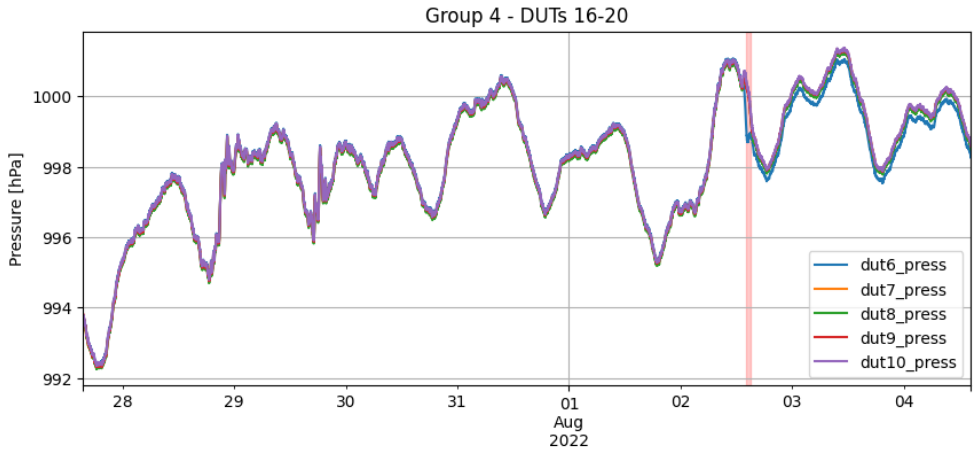


Figure 4.13: Measured pressure of the sensors in Group 4 of Dataset 4 throughout the duration of the entire experiment. The thermal stress interval is highlighted in red.

- Group 6 (DUTs 24-29): No thermal stress was applied; this group served as a reference, with the mean pressure at each time step used to assess the error of sensors in the stressed groups.

Raw data were recorded directly from the sensors, except during thermal stress periods where cubic spline interpolation was used as a data-cleaning method. For example, irregular pressure spikes caused by the setup and shutdown of the high-temperature oven were considered artifacts and excluded from analysis as they did not reflect the sensors' true response.

Figure 4.13 presents a focused analysis of Group 4 sensors behavior during the entire experimental period. The plot shows the measured pressure values of each sensor over time, with the thermal stress interval applied to these devices highlighted in red. This interval represents the one-hour period during which the sensors were exposed to a temperature of -20°C, simulating environmental stress.

After the stress period, a significant deviation is observed in DUT 6, shown in blue, which exhibits a clear drift from the baseline pressure compared to its initial values and the other sensors in the group. This drift suggests that DUT 6 may have experienced a change in its sensitivity or calibration due to the stress exposure. By

contrast, the other sensors in Group 4 show a more stable response over time, indicating resilience to the applied thermal conditions. This observation is crucial because it demonstrates the need for a calibration solution that can specialize on the specific drift behavior of a sensor.

	MAE [hPa]	RMSE [hPa]	SMAPE [%]
Dataset 1	0.3411	0.4088	199.99 %
Dataset 2	0.5958	0.7351	199.77 %
Dataset 3	1.7923	1.9405	200.00 %
Dataset 4	0.0848	0.1059	199.97 %

Table 4.3: Baseline pressure errors for the datasets used in Use Case 2, where the forecast values represent the sensor errors, while the actual values are modeled as a zero-valued time series.

4.2.3 Phase 1: Sensor Manufacturing

As detailed in Section 1.2, Phase 1 corresponds to the mass production stage of the sensors. During this phase, a small random sample of sensors is selected for performance verification. These sensors undergo thermal stress to simulate the soldering process and to characterize their response under such conditions.

Dataset 1 was utilized in this step, as it includes data collected from sensors immediately after soldering. The dataset was divided into training and testing sets, with 10% of the data (8 DUTs) used for training and 90% (72 DUTs) reserved for testing. The first 10% of the data was specifically used for the offline initialization of a global TinyRBF network, while the remaining 90% is used to validate the performance of the initial model. The initialization of the TinyRBF network was conducted using the method described in Section 3.6. A total of 20 neurons were chosen for the network, as this configuration provided an optimal balance between computational complexity and accuracy.

Table 4.4 presents the prediction errors of the generic model for both the training and testing sets. When compared to the baseline error of Dataset 1 (reported in

	MAE [hPa]	RMSE [hPa]	SMAPE [%]
Train set	0.1763	0.2358	82.11 %
Test set	0.1997	0.2603	85.91 %

Table 4.4: Prediction error of the global model initialized using 10% of sensors from Dataset 1.

Table 4.3), the model demonstrates a 41.5% reduction in error on the test set, as measured by the MAE metric.

Figure 4.14 illustrates the comparison between the actual values and the model predictions for the 8 sensors in the training set. The plot highlights that the initialized network focuses on minimizing the overall error across the entire dataset rather than tailoring the predictions to each individual sensor. This global optimization approach is appropriate for the initialization phase, while the sensor-specific adaptation will be the focus of the online procedures carried out in Phases 2 and 3.

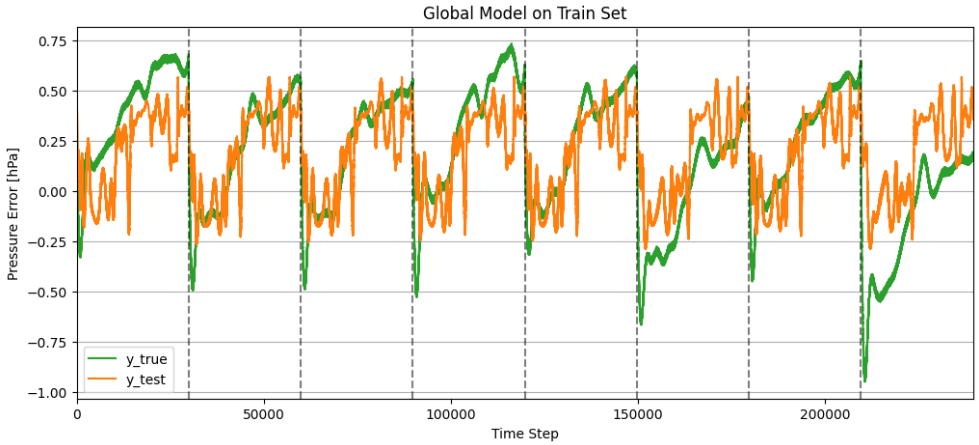


Figure 4.14: Comparison of the true values (green) and model predictions (orange) for the concatenated data from the 8 sensors in the training set.

4.2.4 Phase 2: Motherboard Soldering

Phase 2 simulates the soldering process of a pressure sensor onto the motherboard of the final device. Due to the fact that this step introduces thermal stress to the sensor, Dataset 1, representing the post-thermal stress condition, was consistently used in this phase of the experiment. Conversely to Phase 1, the global model was further specialized to adapt to the drift behavior of each individual sensor. The experimental setup assumes a quasi-laboratory validation scenario, where reference pressure values are frequently available for calibration. Specifically, three different average intervals for reference data availability (t_{ref}) were evaluated: 1 minute, 5 minutes, and 10 minutes. This variation in reference availability allowed for the analysis of the model's ability to adapt under different conditions, imitating practical constraints on real-time recalibration.

t_{ref}	r_{init}	ε	β	η	γ	M	α	W
1 minute	-	0.002	0.75	0.2	2.0	72	0.01	1
5 minutes	-	0.002	0.75	0.4	2.0	72	0.01	1
10 minutes	-	0.002	0.75	0.4	2.0	72	0.01	1

Table 4.5: Best hyperparameters that regulates the evolution of the TinyRBF network over time for each mean reference availability period.

A grid search methodology was employed to identify the optimal hyperparameters for the network, ensuring the best prediction accuracy. Grid search is a methodical approach to hyperparameter tuning that involves exhaustively searching through a specified set of hyperparameter values to find the optimal combination for model performance. The results of the grid search, including the best-performing configurations for each value of t_{ref}, are detailed in Table 4.5. The table outlines key hyperparameters such as the learning rates, distance threshold amplitude and pruning terms.

The performance of the network using these optimized parameters was evaluated through experiments that produced the results presented in Figure 4.15. The reported values were obtained by repeating each experiment ten times with a different random seed, and computing the average. This figure illustrates the maximum

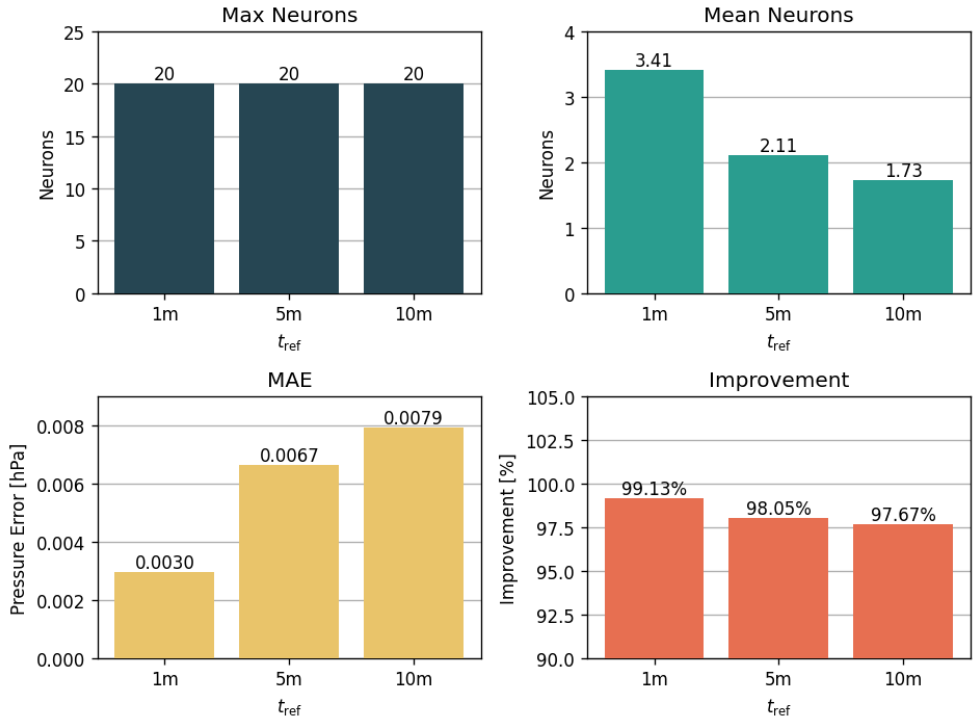


Figure 4.15: Results of applying the TinyRBF approach to Dataset 1 for Phase 2 in terms of maximum and average number of neurons, sensor precision after calibration and error reduction compared to dataset baseline.

and minimum number of neurons used in the network and the corresponding MAE and RMSE values, offering insight into the model's predictive accuracy under different configurations. Furthermore, improvements compared to the Dataset 1 baseline are reported, demonstrating the great error reduction capacity of the model which is always greater than 97%. While the maximum number of neurons is always equal to 20 due to the initialization of the network in the previous phase, on average the model was characterized by a number of RBF nodes ranging from 1.7 to 3.5.

Figure 4.16: Example of application of the TinyRBF approach to DUT 92 of Dataset 1 considering $t_{\text{ref}} = 10$ minutes.

An application example of the on-device learning approach to sensor 92 of Dataset 1 is reported in Figure 4.16. This figure contains three plots containing the comparison between the model prediction and the reference data, the error values of the input

pattern and the distance from the closest node, and the evolution of the number of neurons in the hidden layer of the network. It is possible to notice how, unlike the error threshold, the distance threshold from the closest node is dynamic and adapts to the values found in the input patterns. This behavior is possible thanks to the adaptive distance threshold described in Section 3.4.1, and guarantees a good compromise between complexity and precision of the model. Regarding the number of neurons in the hidden layer of the network, initially the count is 20 due to the initialization performed in Phase 1. Subsequently, the number of nodes decreases drastically due to the stringent pruning parameters ($W = 1$) that lead the model to maintain only the minimum number of significant neurons.

4.2.5 Phase 3: End-User Device

Phase 3 represents the deployment and usage of the device by the end user. The primary objective during this phase is to implement a sensor calibration process that is both seamless from the user's perspective and adaptable to varying operating conditions, ranging from standard use to extreme stress scenarios. Furthermore, it is assumed that the acquisition of reference values for the sensor is a costly process and can only be performed infrequently. To address these constraints, Datasets 2, 3, and 4 were utilized in this phase. These datasets provide data from sensors operating under both normal and stressed conditions, enabling the evaluation of the calibration approach across diverse scenarios. In addition, they facilitate a comparison between two sensors differing in precision and reliability, providing valuable information on the adaptability and robustness of the proposed calibration method. In detail, it is discussed the application of the TinyRBF algorithm for error compensation of the LPS22HH pressure sensor, specifically addressing performance under normal conditions and extreme thermal and pressure stresses. Finally, it is explored the drift reduction method applied to the high-precision LPS22DF barometric sensor, which was exposed to five different types of thermal stress for testing.

Assuming that sensor data are sampled at a regular interval, at each sampling step, the network performs inference or training, as described in Sections 3.3 and 3.4, respectively. Model updates are possible only when a reference value is available,

enabling calculation of the prediction error. To make experiments realistically valid, the reference values appear with a probability of p_{ref} at each step, which means that, on average, learning can occur every t_{ref} seconds. The relationship between p_{ref} and t_{ref} is governed by the following formula:

$$t_{\text{ref}} = \frac{t_{\text{sample}}}{p_{\text{ref}}} \tag{4.4}$$

where p_{ref} is the probability at each time instant t to have a reference value $\hat{y}$, t_{sample} is the sampling period (in these experiments, its value is 30 seconds), and t_{ref} is the average interval between two references. In particular, the values selected for t_{ref} are: 1 hour, 3 hours, 6 hours, 12 hours, 1 day, 2 days, and 3 days. These values correspond to the following probabilities p_{ref}, respectively: 0.008333 %, 0.002778 %, 0.001389 %, 0.000694 %, 0.000347 %, 0.000174 %, and 0.000116 %.

To ensure a robust evaluation of the model's performance across all datasets, 10 different random seeds were used during the validation process. This approach provides a more comprehensive assessment of the model's generalization capabilities and reduces the dependence on favorable reference sequences. By averaging the results obtained across these 10 runs, the influence of random factors was minimized, leading to more reliable and consistent performance metrics for each dataset.

In addition, the following experiments used a grid search methodology, which is a systematic approach used to identify optimal hyperparameters for a model by exhaustively evaluating combinations of predefined parameter values. For each set of hyperparameters, the model performance is evaluated using the entire dataset and the ten different random seeds, and the configuration that produces the best results is selected. This approach ensures thorough exploration of the parameter space to achieve the best result in terms of model accuracy.

In all the experiments conducted with Datasets 2-4, the TinyRBF network used the instantaneous atmospheric pressure and temperature readings from the LPS22HH and LPS22DF sensors as its only two inputs. The network's single output represented the instantaneous prediction of the sensor error. By adding this predicted error to the measured pressure value, the model effectively compensated for the device drift, ensuring improved accuracy in the sensor's readings.

Dataset 2

The optimal hyperparameters for Dataset 2 were determined using a grid search methodology, and the resulting configurations are summarized in Table 4.6.

t_{ref}	r_{init}	ε	β	η	γ	M	α	W
1 hour	2.5	0.02	0.75	0.4	2.0	1440	0.01	1
3 hours	2.5	0.02	0.75	0.6	2.0	1440	0.01	1
6 hours	2.5	0.02	0.75	0.6	1.8	1440	0.01	1
12 hours	2.5	0.02	0.75	0.6	2.0	1440	0.01	1
1 day	2.5	0.02	0.75	0.6	2.0	1440	0.01	1
2 days	2.5	0.02	0.75	0.4	0.6	1440	0.01	1
3 days	2.5	0.02	0.75	0.6	1.8	1440	0.01	1

Table 4.6: Best hyperparameters of the learning process for Dataset 2 in Phase 3.

Table 4.7 presents the experimental results in terms of MAE, RMSE and SMAPE after the application of the TinyRBF approach for online calibration of sensor LPS22HH. Considering a base MAE of the dataset equal to 0.5958 hPa, the on-device learning methodology is able to obtain an error after the compensation starting from 0.2957 hPa up to 0.0337 hPa, based on the different values of t_{ref}. These values correspond to an error reduction percentage varying between 94.35%, when references are more frequent, and 50.37%, when the network update occurs more rarely.

Figure 4.17 reports a visual representation of the results in terms of maximum and mean neurons, MAE, and error reduction with respect to dataset baseline. The higher number of nodes, both maximum and average, observed for $t_{\text{ref}} = 2$ days is attributed to the reduced value of the distance threshold amplitude hyperparameter γ. A lower threshold value allows the network to allocate new nodes more frequently, resulting in an increased node count.

Table 4.8 reports the complexity results, detailing both the maximum and average number of nodes allocated during online learning and the correspondents maximum and mean memory footprint required to store the parameters of the network across different reference availability probabilities p_{ref}. The maximum number of

t_{ref}	MAE [hPa]	RMSE [hPa]	SMAPE [%]	Error Reduction [%]
1 hour	0.0337	0.0562	12.11 %	94.35 %
3 hours	0.0511	0.0873	15.28 %	91.42 %
6 hours	0.0656	0.1052	17.79 %	88.99 %
12 hours	0.0943	0.1511	24.12 %	84.17 %
1 day	0.1607	0.2391	39.51 %	73.03 %
2 days	0.2449	0.3277	68.82 %	58.89 %
3 days	0.2957	0.3665	78.89 %	50.37 %

Table 4.7: Experimental error results for Dataset 2 in Phase 3 for each value of t_{ref} and error reduction with respect to dataset baseline.

t_{ref}	N_{max}	N_{avg}	Max Memory	Mean Memory
1 hour	7	1.97	144 bytes	43.4 bytes
3 hours	6	1.45	124 bytes	33.0 bytes
6 hours	5	1.62	104 bytes	36.5 bytes
12 hours	2	1.11	44 bytes	26.2 bytes
1 day	2	1.06	44 bytes	25.3 bytes
2 days	7	2.12	144 bytes	46.3 bytes
3 days	1	0.86	24 bytes	21.3 bytes

Table 4.8: Network complexity results for Dataset 2 in Phase 3, where N_{max} and N_{avg} are respectively the max and mean number of nodes during the learning process.

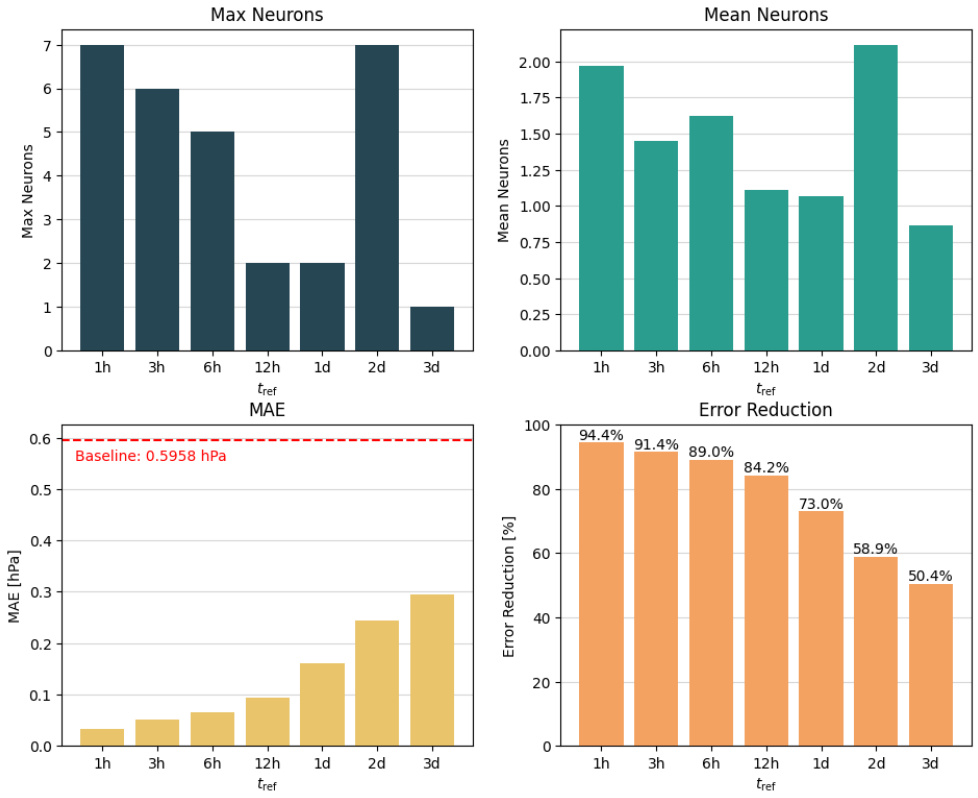


Figure 4.17: Experimental results in terms of maximum and average number of neurons, Mean Absolute Error (MAE), and error reduction with respect to dataset baseline for Dataset 2 in Phase 3.

nodes allocated in all experiments was 7, corresponding to a memory footprint of only 1144 bytes. In general, the number of allocations was very small, and on average the topologies consisted of a number of nodes between 0.86 and 2.12.

With a reference interval of $t_{\text{ref}} = 12$ hours, error reduction reached 84.17% with a maximum of 2 RBF units, corresponding to a memory footprint of 44 bytes and an inference computational complexity of $2C_{\text{exp}} + 21$ FLOPs, computed as in 3.22.

Dataset 3

The best-performing hyperparameters for Dataset 3, identified through grid search, are presented in Table 4.9.

t_{ref}	r_{init}	ε	β	η	γ	M	α	W
1 hour	100.0	0.02	0.75	0.2	0.4	1440	0.01	150
3 hours	100.0	0.02	0.75	0.2	0.2	1440	0.01	25
6 hours	100.0	0.02	0.75	0.3	0.2	1440	0.01	25
12 hours	100.0	0.02	0.75	0.3	0.2	1440	0.01	25
1 day	100.0	0.02	0.75	0.6	0.4	1440	0.01	25
2 days	100.0	0.02	0.75	0.6	0.4	1440	0.01	25
3 days	100.0	0.02	0.75	0.6	0.8	1440	0.01	1

Table 4.9: Best hyperparameters for the learning process of Dataset 3 in Phase 3.

Due to the intensive pressure and temperature stresses applied to the sensors, the base MAE of the dataset is equal to 1.7923 hPa. Starting from this value, the application of the TinyRBF network brought the mean absolute error values between 1.0784 hPa and 0.0832 hPa, as reported in Table 4.10. The error reduction is high (80-95%) when a reference is frequently available, while it is lower but still effective (40-70%) for infrequent calibrations.

The graphs in Figure 4.18 show the maximum and average number of neurons, the average absolute error, and the improvement in sensor accuracy compared to dataset baseline. It can be noted that, even when the reference value is available on average every two days, the measurement error is halved by using a maximum of 5

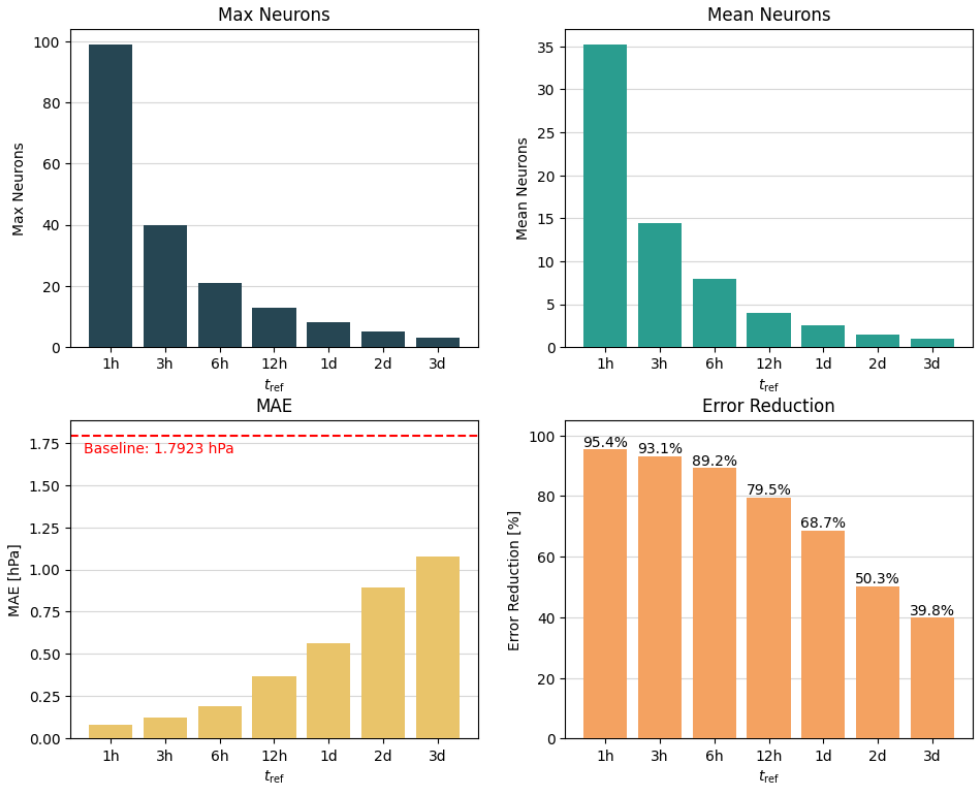


Figure 4.18: Experimental results in terms of maximum and average number of neurons, Mean Absolute Error (MAE), and error reduction with respect to dataset baseline for Dataset 3 in Phase 3.

t_{ref}	MAE [hPa]	RMSE [hPa]	SMAPE [%]	Error Reduction [%]
1 hour	0.0832	0.147941	8.61 %	95.36 %
3 hours	0.1236	0.207254	13.10 %	93.10 %
6 hours	0.1934	0.298314	19.53 %	89.21 %
12 hours	0.3674	0.520458	37.41 %	79.50 %
1 day	0.5615	0.725964	55.83 %	68.67 %
2 days	0.8911	1.056729	92.12 %	50.28 %
3 days	1.0784	1.240197	113.43 %	39.83 %

Table 4.10: Experimental error results for Dataset 3 in Phase 3 for each value of t_{ref} and error reduction with respect to dataset baseline.

hidden neurons, which correspond to 104 bytes of memory used to store all model parameters.

Table 4.11 details the network complexity in terms of maximum and average number of RBF nodes and their memory usage. Although the exponential decrease

t_{ref}	N_{max}	N_{avg}	Max Memory	Mean Memory
1 hour	99	35.25	1984 bytes	709.0 bytes
3 hours	40	14.38	804 bytes	291.7 bytes
6 hours	21	7.94	424 bytes	162.7 bytes
12 hours	13	3.99	264 bytes	83.8 bytes
1 day	8	2.52	164 bytes	54.3 bytes
2 days	5	1.41	104 bytes	32.2 bytes
3 days	3	0.97	64 bytes	23.4 bytes

Table 4.11: Network complexity results for Dataset 3 in Phase 3, where N_{max} and N_{avg} are respectively the max and mean number of nodes during the learning process.

in the maximum and average number of neurons as t_{ref} increases, high values of maximum network complexity were obtained for frequent calibrations. This behavior is justified by the characteristics of the dataset and the challenging compensation task submitted to the on-device learning model.

Dataset 4

Grid search was employed to systematically explore and identify the optimal hyperparameters that minimize the sensor error. The best hyperparameters found are reported in Table 4.12. In addition, the final results for each sensor group were calcu-

t_{ref}	r_{init}	ε	β	η	γ	M	α	W
1 hour	20.0	0.02	0.75	0.3	2.0	1440	0.01	25
3 hours	20.0	0.02	0.75	0.4	2.0	1440	0.01	1
6 hours	20.0	0.02	0.75	0.4	1.8	1440	0.01	25
12 hours	20.0	0.02	0.75	0.5	2.0	1440	0.01	1
1 day	20.0	0.02	0.75	0.5	2.0	1440	0.01	1
2 days	20.0	0.02	0.75	0.4	1.2	1440	0.01	1
3 days	20.0	0.02	0.75	0.4	1.4	1440	0.01	1

Table 4.12: Best hyperparameters for the learning process of Dataset 4 in Phase 3.

lated by averaging the evaluation metrics, such as MAE and RMSE, across all sensors in the group.

Although the baseline mean absolute error of the dataset is very low at 0.0848 hPa due to the high precision of the LPS22DF sensor, the on-device learning model was able to reduce this error from 43% to 84.7%, as detailed in Table 4.13. In this way, the compensated sensor error was found to be between 0.0122 hPa and 0.0481 hPa, in relation to the average period of availability of a reference value for model update.

Table 4.14 reports the detail of the measurement error reduction for each sensor group. Although group 5 is characterized by the highest starting error, it presents the highest percentages of accuracy improvement, ranging from 53.1% to 92.5%.

Figure 4.19 visually reports the maximum and average number of neurons allocated by the TinyRBF network, the average absolute error obtained through compensation, and the percentage improvement compared to the baseline of the dataset. The graphs show the values for each average recalibration frequency t_{ref} and specifically for each sensor group, with the aim of highlighting the results at different thermal

t_{ref}	MAE [hPa]	RMSE [hPa]	SMAPE [%]	Error Reduction [%]
1 hour	0.0122	0.0362	27.30 %	84.74 %
3 hours	0.0159	0.0445	31.99 %	80.01 %
6 hours	0.0180	0.0473	35.04 %	77.52 %
12 hours	0.0206	0.0513	42.36 %	74.49 %
1 day	0.0279	0.0582	60.32 %	66.15 %
2 days	0.0413	0.0686	92.00 %	50.69 %
3 days	0.0481	0.0733	109.08 %	43.09 %

Table 4.13: Experimental error results for Dataset 4 in Phase 3 for each value of t_{ref} and error reduction with respect to dataset baseline.

t_{ref}	**Group 1**	**Group 2**	**Group 3**	**Group 4**	**Group 5**
Base MAE [hPa]	0.0738	0.0768	0.0658	0.0784	0.1401
1 hour	82.9%	87.1%	79.3%	82.0%	92.5%
3 hours	77.2%	83.0%	73.3%	76.4%	90.2%
6 hours	75.4%	79.4%	70.9%	73.0%	88.9%
12 hours	71.8%	75.9%	68.8%	70.1%	86.0%
1 day	67.3%	65.5%	58.1%	62.4%	77.5%
2 days	53.9%	51.2%	42.8%	43.0%	62.5%
3 days	45.8%	43.7%	36.7%	36.2%	53.1%

Table 4.14: Mean error reduction for Dataset 4 in Phase 3 with respect to initial MAE of each sensor group for every value of average reference periodicity.

stresses. It can be noted that the error grows with a similar linear behavior with the average reference periodicity, despite the non-linear decline in periodicity values.

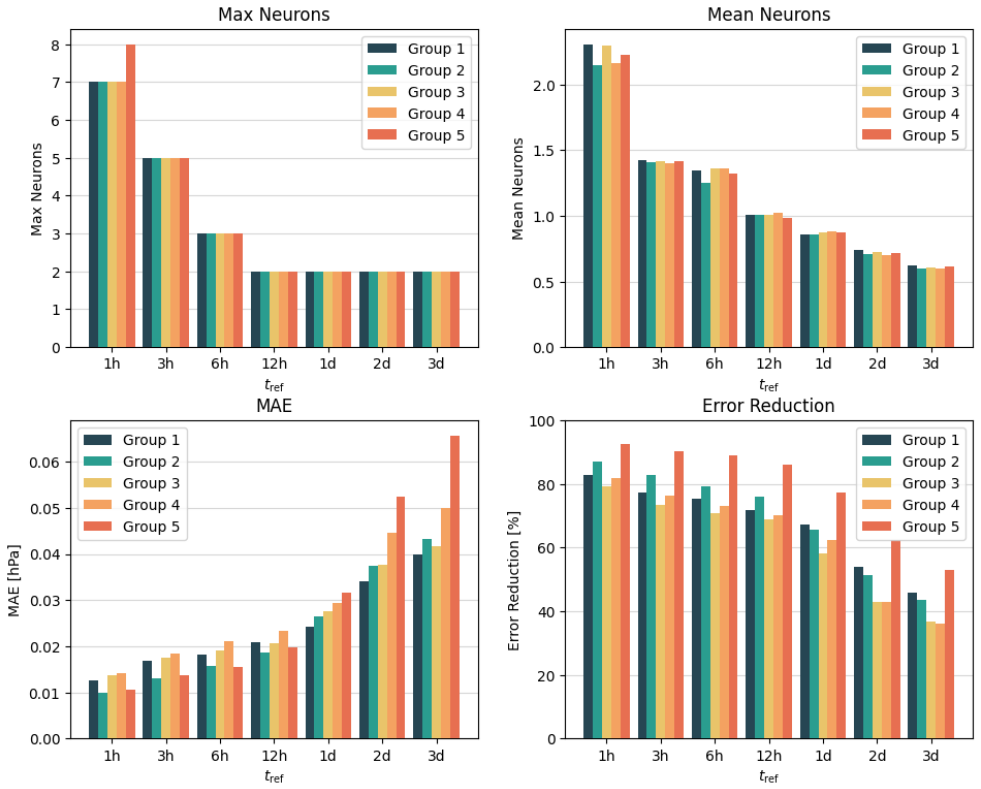


Figure 4.19: Experimental results in terms of maximum and average number of neurons, Mean Absolute Error (MAE), and error reduction with respect to dataset baseline for each sensor group in Dataset 4 in Phase 3.

Regarding the number of network parameters, Table 4.15 shows a global maximum of 8 neurons, with an average of 1.17 RBF nodes. Assuming that parameters are represented in 32-bit floating-point format, the memory footprint of the model, computed using Equation 3.21, averages 27.4 bytes and reaches a maximum of 164 bytes.

t_{ref}	N_{max}	N_{avg}	Max Memory	Mean Memory
1 hour	8	2.228	164 bytes	48.6 bytes
3 hours	5	1.411	104 bytes	32.2 bytes
6 hours	3	1.328	64 bytes	30.6 bytes
12 hours	2	1.008	44 bytes	24.2 bytes
1 day	2	0.867	44 bytes	21.4 bytes
2 days	2	0.717	44 bytes	18.4 bytes
3 days	2	0.609	44 bytes	16.2 bytes

Table 4.15: Network complexity results for Dataset 4 in Phase 3, where N_{max} and N_{avg} are respectively the max and mean number of nodes during the learning process.

4.3 On-Field Correction of Environmental Data

This section presents and analyzes the experimental results for *Use Case 3*, as outlined in Section 1.2.3. The sensors utilized for field experimentation in environmental data collection are detailed in Section 4.3.1. The hardware setup is outlined in Section 4.3.2, while the dataset collected during the experimentation is described in Section 4.3.3. Finally, the analysis of the results is presented in Section 4.3.4.

4.3.1 Sensors

The field experimentation for environmental data collection utilized a variety of sensors, each selected for its ability to measure specific environmental parameters. These sensors, detailed below, include both cutting-edge and widely-used devices to ensure comprehensive data acquisition and facilitate comparisons:

1. **LPS22DF**[3]: A low-power and high-precision MEMS barometric pressure sensor by STMicroelectronics known for its accuracy and stability under varying environmental conditions. It provides precise pressure and temperature readings essential for atmospheric data collection.

2. **SHT40AD1B**[4]: A state-of-the-art digital sensor by Sensirion designed for high-accuracy temperature and humidity measurements. Its compact design and low power consumption make it ideal for field applications.

3. **STTS22H**[5]: A low-voltage and ultra-low-power digital temperature sensor by STMicroelectronics offering good precision and reliability, even in challenging environmental conditions.

4. **BME280**[6]: A widely-used environmental sensor capable of measuring temperature, humidity, and pressure by Bosch. It is valued for its versatility and

[3] https://www.st.com/en/mems-and-sensors/lps22df.html
[4] https://sensirion.com/products/catalog/SHT40/
[5] https://www.st.com/en/mems-and-sensors/stts22h.html
[6] https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/

integrated measurement capabilities.

5. **DHT22**[7]: A digital sensor by DFRobot providing temperature and humidity readings with reasonable accuracy and cost-effectiveness, commonly employed in various field applications.

6. **DHT11**[8]: Similar to the DHT22 but with a simpler design, this sensor by Seeed Studio provides basic temperature and humidity data, making it suitable for less demanding applications.

In addition to these sensors, the measurements were integrated with public data provided by the *ARPAE* "S. Pancrazio" weather station[9]. This station integrates the *HYT 221*, a high-quality temperature and relative humidity sensor, that was selected as a reference device, ensuring the reliability of the collected data.

Altogether, these sensors enabled the collection of a diverse range of environmental data, ensuring that the field experimentation captured variations across different conditions and sensor types. The accuracy specifications listed in Table 4.16 provide a detailed comparison of the performance capabilities of each sensor, as documented by their respective manufacturers. Based on the absolute ratings of the devices, the LPS22DF was selected as the reference sensor for barometric pressure measurements, while the HYT 221 was chosen as the reference for temperature and humidity measurements.

4.3.2 Experimental Setup

The experimental setup for environmental data collection was designed to ensure robust and reliable measurements in real-world, outdoor conditions. The setup utilized the *NUCLEO-WL55JC*[10] development board by STMicroelectronics, which

[7] https://wiki.dfrobot.com/DHT22_Temperature_and_humidity_module_SKU_SEN0137

[8] https://wiki.seeedstudio.com/Grove-TemperatureAndHumidity_Sensor/

[9] https://www.arpae.it/it/temi-ambientali/meteo/dati-e-osservazioni/dati-in-tempo-reale/grafici-per-stazione?s=2286

[10] https://www.st.com/en/evaluation-tools/nucleo-wl55jc.html

Sensor	Pressure	Temperature	Relative Humidity
LPS22DF	±0.2 hPa	±1.5 °C	
SHT40AD1B		±0.2 °C	±1.8 %RH
STTS22H		±0.5 °C	
BME280	±1.0 hPa	±0.5 °C	±3.0 %RH
DHT22		±0.5 °C	±2.0 %RH
DHT11		±2.0 °C	±5.0 %RH
HYT221		±0.2 °C	±1.8 %RH

Table 4.16: Measurement accuracy of the various sensors used for collecting environmental data as specified in the official manuals of device manufacturers.

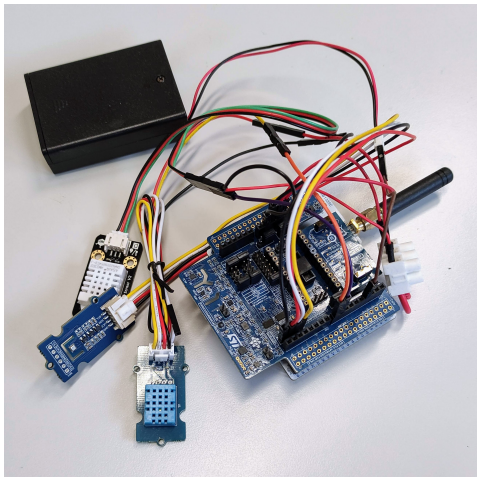
served as the primary platform for the experiment. This board is equipped with the STM32WL55JC microcontroller running up to 48 MHz, a dual-core device that integrates an ARM Cortex-M4 core for application processing and a Cortex-M0+ core dedicated to low-power tasks and communication management. The MicroController Unit (MCU) is equipped with 256 KB of flash memory and 64 KB of SRAM, providing ample storage for data processing, sensor readings, and program execution.

To expand the NUCLEO-WL55JC board's functionality, the *X-NUCLEO-IKS4A1*[11] expansion board by STMicroelectronics was used. This expansion board, specifically designed for environmental sensing, provides an easy interface for connecting various sensors to the development board. In particular, it features the high-precision LPS22DF pressure sensor, the high-accuracy and ultra-low-power SHT40AD1B relative humidity and temperature sensor (by Sensirion), and the ultralow-power STTS22H temperature sensor. Additionally, three external sensors have been added to the setup, namely the BME280 pressure, temperature and humidity sensor and the DHT22 and DHT11 temperature and relative humidity sensors. All sensors except DHTs, were connected to the development board via the standard interface I2C for consistent communication and data acquisition. DHT22 and DHT11 sensors were wired to a General Purpose Input Output (GPIO) pin, since they are using a non standard digital

[11] `https://www.st.com/en/ecosystems/x-nucleo-iks4a1.html`

communication protocol to transmit temperature and humidity readings.

The entire hardware setup was powered by 3-AA rechargeable battery pack, ensuring portability and the ability to operate autonomously for extended periods. All components were enclosed in a waterproof box, designed specifically for outdoor use. The enclosure was built to protect the hardware from moisture and other environmental hazards, while still allowing air to circulate to ensure proper sensor function and accurate readings. This setup allowed for continuous data collection over the duration of the experiment, enabling the analysis of sensor performance under real-world conditions.



(a) Hardware setup featuring the NUCLEO-WL55JC development board, the X-NUCLEO-IKS4A1 expansion board, external sensors (BME280, DHT22, and DHT11), and a 3-AA battery pack for power supply.

(b) Deployment in field of the air-permeable, waterproof data collection node near the ARPAE "San Pancrazio" weather station in Parma, Italy.

Figure 4.20: Configuration of the data collection node and its installation in the field.

Figure 4.20 depicts the hardware setup of the data collection node, featuring the NUCLEO-WL55JC board, X-NUCLEO-IKS4A1 expansion, external sensors (BME280, DHT22, DHT11), and its waterproof yet air-permeable installation near the ARPAE "San Pancrazio" weather station in Parma, Italy.

The LoRaWAN protocol was employed to transmit sensor data every 10 minutes, ensuring low power consumption and reliable communication over long distances. The Things Network (TTN) service was utilized to route the data from LoRaWAN gateways to our application. The application, implemented as an MQTT client, subscribed to the TTN uplink messages and stored the received data in a CSV file for subsequent analysis. This setup provided an efficient and scalable solution for collecting and archiving environmental data in near real-time.

4.3.3 Dataset

The dataset used for the environmental data analysis includes measurements of barometric pressure, temperature, and humidity collected from the selected sensors: LPS22DF, SHT40AD1B, STTS22H, BME280, DHT22, and DHT11. In addition, the HYT 221, integrated within a high-quality weather station, served as the reference device for temperature and humidity validation.

Measurements were collected from the selected sensors over a continuous period of 52 days, 10 hours, and 30 minutes. Data collection began on October 10, 2024, at 18:20 UTC and concluded on December 2, 2024, at 04:50 UTC, resulting in a total of 7552 samples. Measurements were recorded simultaneously from all devices, enabling consistent comparisons of sensor performance.

Figures 4.21, 4.22, and 4.23 provide an overview of the pressure, temperature, and humidity values recorded throughout the entire experimental period.

Figure 4.24 presents the mean absolute error for each sensor's readings compared to the reference values, providing a comparison of sensor performance at the start of the experiment. Table 4.17 shows the overall mean absolute error for each sensor that measure more than one environmental parameter, calculated by combining the errors from pressure, temperature, and humidity readings, and compared to reference values.

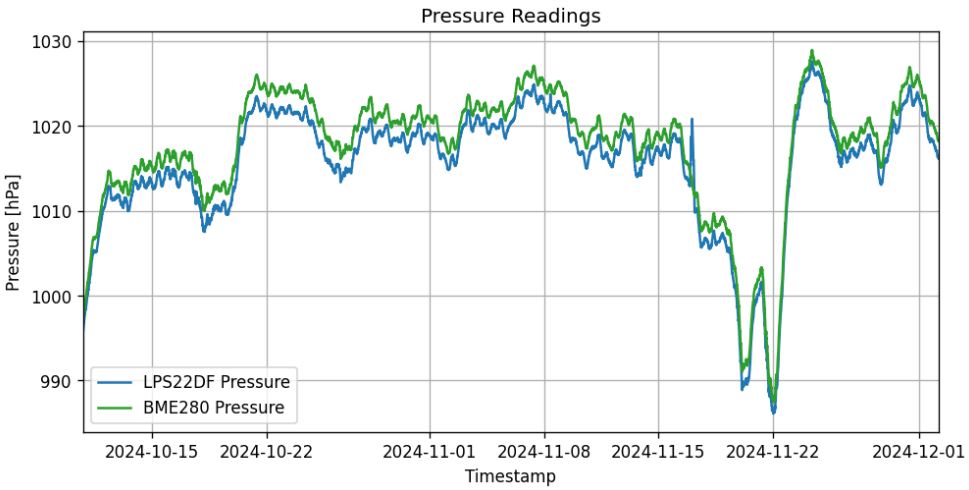


Figure 4.21: Overview of sensor readings for barometric pressure collected from the LPS22DF and BME280 sensor.

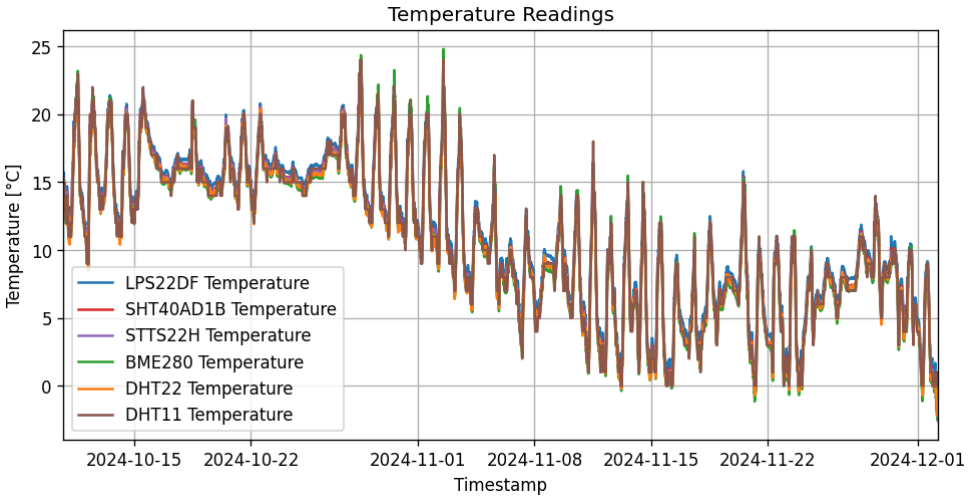


Figure 4.22: Overview of sensor readings for temperature collected from the LPS22DF, SHT40AD1B, STTS22H, BME280, DHT22, and DHT11 sensors.

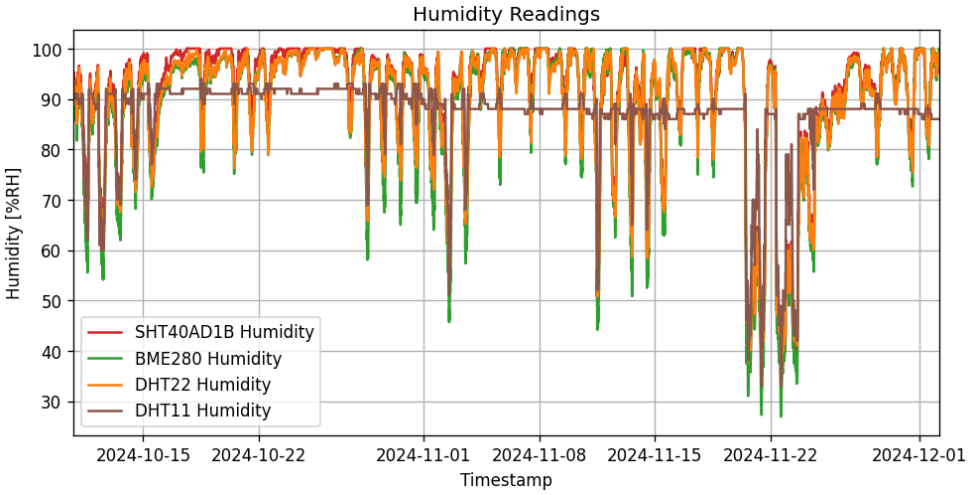


Figure 4.23: Overview of sensor readings for relative humidity collected from the SHT40AD1B, BME280, DHT22, and DHT11 sensors.

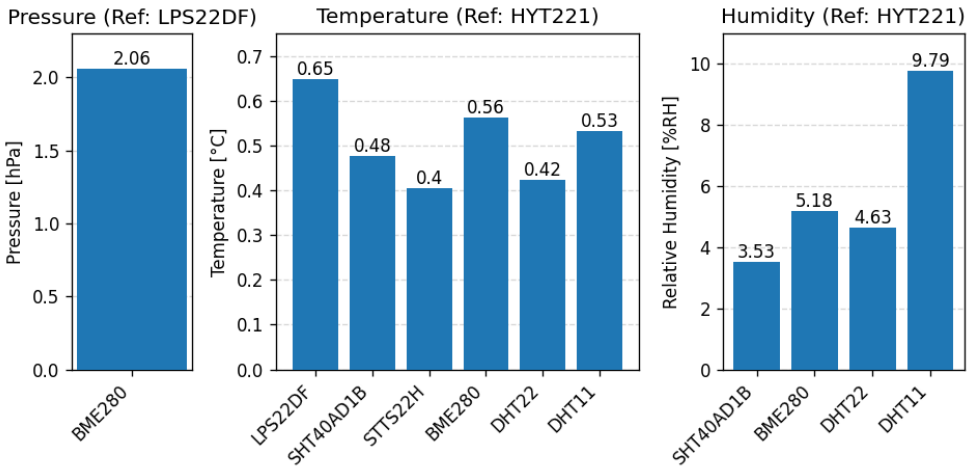


Figure 4.24: Baseline error (MAE) for each sensor across the three variables (pressure, temperature, and humidity).

	MAE [hPa,°C,%RH]	RMSE [hPa,°C,%RH]	SMAPE [%]
BME280	2.5989	4.0468	6.37 %
SHT40AD1B	2.0027	3.4912	6.91 %
DHT22	2.5271	4.1868	7.85 %
DHT11	5.1606	7.5056	11.36 %

Table 4.17: Aggregated MAE across all three variables (pressure, temperature, and humidity) for each sensor that measure more than one environmental parameter.

4.3.4 Results

Considering that sensor data is sampled at regular intervals, at each sampling step the network either performs inference or training, as detailed in Sections 3.3 and 3.4, respectively. Model updates occur only when a reference value is available, enabling the calculation of the prediction error. To ensure realistic experimental conditions, reference values are provided with a probability of p_{ref} at each step, meaning that, on average, learning can happen every t_{ref} seconds.

The relationship between p_{ref} and t_{ref} is described by the following equation:

$$t_{\text{ref}} = \frac{t_{\text{sample}}}{p_{\text{ref}}} \tag{4.5}$$

where p_{ref} is the probability of receiving a reference value $\hat{y}$ at each time instant t, t_{sample} is the sampling period (10 minutes in these experiment), and t_{ref} is the average interval between two references. For these experiments, the selected values of t_{ref} are: 1 hour, 6 hours, 12 hours, 1 day, 2 days, 3 days, 5 days, and 7 days. These correspond to the following probabilities p_{ref}: 0.166667%, 0.027778%, 0.013889%, 0.006944%, 0.003472%, 0.002315%, 0.001389%, 0.000992%, respectively.

To ensure a robust evaluation of the model's performance across all configurations, 20 different random seeds were employed during the validation process. This approach enabled a more comprehensive assessment of the model's generalization capabilities and reduced dependency on specific reference sequences. By averaging the results obtained across these 20 runs, the impact of random variations was minimized, resulting in more reliable and consistent performance metrics for each dataset.

Subsequent experiments utilized a grid search methodology, a systematic approach to identify the optimal hyperparameters for the model by exhaustively evaluating all combinations of predefined parameter values. For each hyperparameter configuration, the model's performance was assessed across the entire dataset using the different random seeds. The configuration yielding the best results was selected. This method ensures a comprehensive exploration of the parameter space, optimizing model accuracy.

The TinyRBF network approach was compared to the prediction capabilities of a *naive model* that simply predicts the last received reference value. This comparison was conducted to evaluate whether the added complexity of the TinyRBF network is justified and necessary to address the task effectively. The naive model operates by updating its prediction only when a new reference value becomes available, holding that value constant until the next update. While simplistic, this model serves as a baseline to assess the performance improvements provided by the TinyRBF network in handling sensor drift and error compensation.

The topology of the TinyRBF networks employed in this experiment varied depending on the sensor being calibrated to accommodate the specific number of input and output variables. Specifically, for the BME280 sensor, the network is designed with three input nodes corresponding to the measured values of pressure, temperature, and relative humidity. Similarly, it features three output nodes, each predicting the error for one of the three variables (pressure, temperature, and humidity). This architecture ensures a tailored correction for each parameter simultaneously. On the other hand, for the SHT40AD1B, DHT22, and DHT11 sensors, the network has two input nodes corresponding to the measured values of temperature and relative humidity. The output layer consists of two nodes, each dedicated to predicting the error for temperature and humidity. This simpler topology aligns with the reduced set of variables measured by these sensors.

Figures 4.25, 4.26, 4.27 and 4.28 show a performance comparison between the TinyRBF network (blue) and the naive model (orange) for error reduction in BME280, SHT40AD1B, DHT22 and DHT11 sensors, respectively, with errors aggregated across pressure, temperature, and relative humidity. The red dashed line represents the ag-

gregated baseline MAE for the sensor, as reported in Table 4.17. The red percentage indicates the reduction in error achieved by the TinyRBF network when compared to the baseline, while the blue one illustrates the difference in performance between the naive model and the TinyRBF network. This comparison highlights the effectiveness of the TinyRBF network in improving the accuracy of sensor measurements.

The detailed accuracy and complexity results of the TinyRBF calibration on the BME280, SHT40AD1B, DHT22 and DHT11 sensors are reported in Tables 4.18, 4.19, 4.20 and 4.21, respectively. These tables report the maximum and average number of neurons allocated in the hidden layer and the sensor error in terms of MAE, RMSE and SMAPE, across the different recalibration periods t_{ref}.

Overall, the on-device learning approach consistently achieves positive sensor error reduction. The improvement is particularly notable for the BME280 sensor, with error reductions ranging from 33% to 80% with respect to dataset baseline MAE. For the SHT40AD1B sensor, while the impact is less pronounced, error reductions still range from 1% to 61%. In all cases, the error growth is sublinear with respect to the value of t_{ref} which grows superlinearly.

The TinyRBF network consistently outperforms the naive model across all reference intervals, with lower MAE values. Although for very frequent reference values ($t_{\text{ref}} = 1$ hour) the naive model's performance is equivalent to those of the TinyRBF network, its error reduction capability diminishes significantly as the intervals increase. The performance gap between the TinyRBF network and the naive model narrows in percentage for larger t_{ref}, but the TinyRBF network remains consistently better. It is important to underline that in some cases, as for the SHT40AD1B sensor, the naive model results in worsening the measurement accuracy of the sensor. On the contrary, the TinyRBF network, even if in some cases with little effectiveness, has a positive contribution to the reduction of measurement error.

In terms of complexity, the number of RBF nodes employed in the hidden layer is limited, with a maximum of 52 neurons allocated, which corresponds to 419 parameters and 1676 bytes of memory. For larger recalibration intervals, the average number of neurons was very low. For instance, in the case of a recalibration every week, the average number of nodes was 1.9 across all tested sensors, with an average reduction

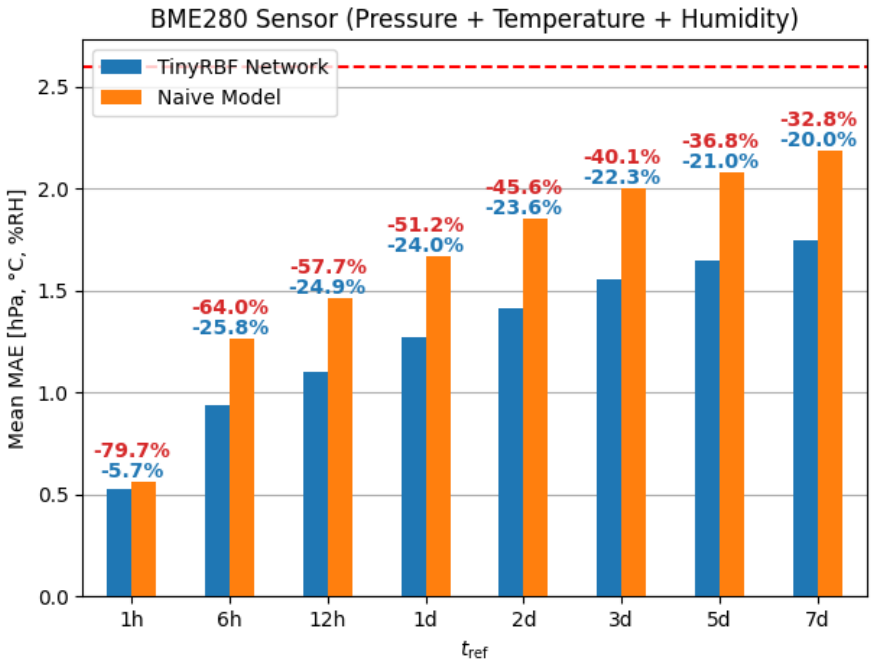


Figure 4.25: Performance comparison between the TinyRBF network (blue) and the naive model (orange) for BME280 sensor.

t_{ref}	N_{max}	N_{avg}	MAE	RMSE	SMAPE [%]
1 hour	52	24.45	0.5279	1.4631	32.39 %
6 hours	50	23.54	0.9368	2.0396	53.89 %
12 hours	29	11.90	1.0999	2.2639	60.43 %
1 day	22	7.91	1.2683	2.5018	67.92 %
2 days	16	5.69	1.4134	2.6763	74.34 %
3 days	13	4.10	1.5578	2.8901	79.03 %
5 days	8	2.44	1.6432	2.9469	83.96 %
7 days	6	2.02	1.7471	3.1041	94.15 %

Table 4.18: Detailed precision and complexity results of TinyRBF network application for BME280 sensor calibration, where N_{max} and N_{avg} are respectively the max and mean number of nodes during the learning process.

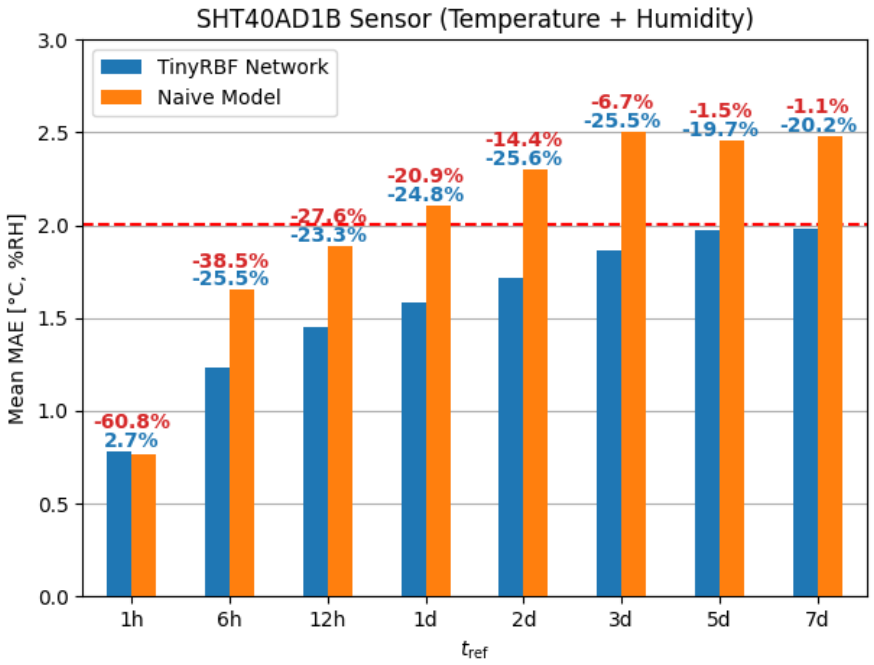


Figure 4.26: Performance comparison between the TinyRBF network (blue) and the naive model (orange) for SHT40AD1B sensor.

t_{ref}	N_{max}	N_{avg}	MAE	RMSE	SMAPE [%]
1 hour	18	7.78	0.7842	1.9223	65.67 %
6 hours	28	13.51	1.2311	2.4077	99.49 %
12 hours	18	7.51	1.4499	2.6255	110.86 %
1 day	23	9.77	1.5836	2.8783	118.64 %
2 days	14	5.23	1.7145	2.9645	125.17 %
3 days	11	3.55	1.8675	3.1225	129.02 %
5 days	7	2.17	1.9727	3.2447	137.13 %
7 days	6	1.84	1.9812	3.2946	141.25 %

Table 4.19: Detailed precision and complexity results of TinyRBF network application for SHT40AD1B sensor calibration, where N_{max} and N_{avg} are respectively the max and mean number of nodes during the learning process.

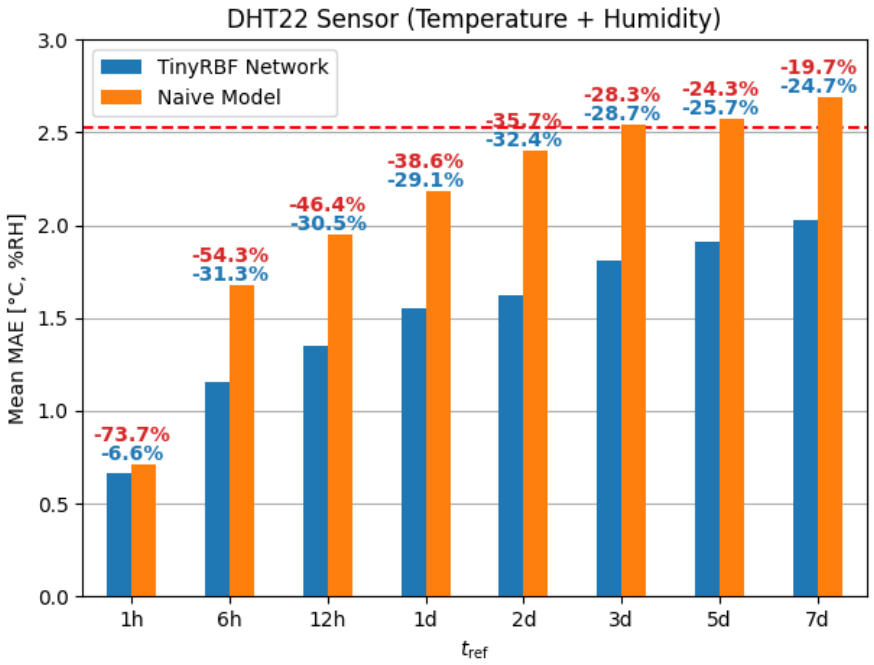


Figure 4.27: Performance comparison between the TinyRBF network (blue) and the naive model (orange) for DHT22 sensor.

t_{ref}	N_{max}	N_{avg}	MAE	RMSE	SMAPE [%]
1 hour	49	28.23	0.6658	1.6115	50.98 %
6 hours	45	20.90	1.1543	2.2229	82.92 %
12 hours	24	10.99	1.3538	2.4670	93.53 %
1 day	18	7.12	1.5515	2.7146	102.58 %
2 days	12	4.97	1.6254	2.7766	106.69 %
3 days	11	3.97	1.8128	3.0550	111.44 %
5 days	8	2.35	1.9130	3.1904	114.19 %
7 days	6	1.97	2.0302	3.3850	123.20 %

Table 4.20: Detailed precision and complexity results of TinyRBF network application for DHT22 sensor calibration, where N_{max} and N_{avg} are respectively the max and mean number of nodes during the learning process.

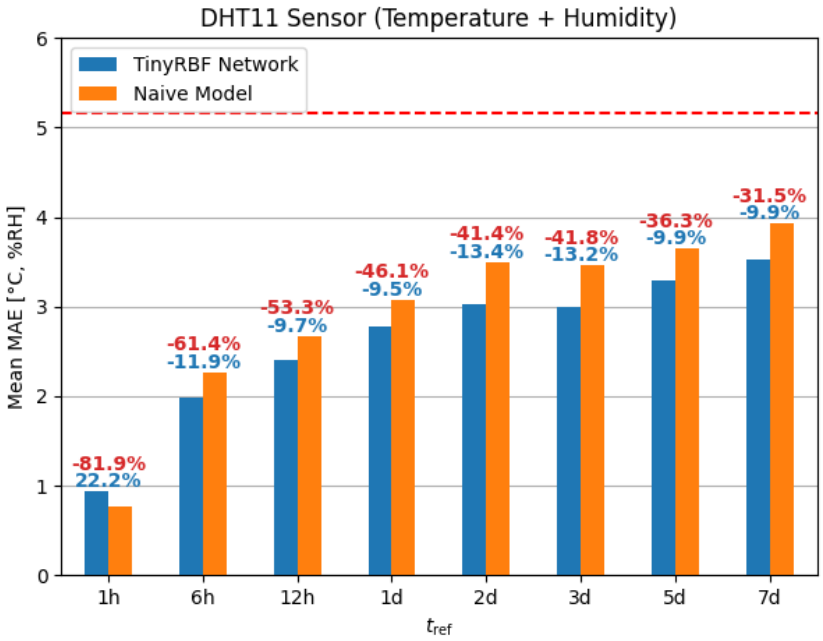


Figure 4.28: Performance comparison between the TinyRBF network (blue) and the naive model (orange) for DHT11 sensor.

t_{ref}	N_{max}	N_{avg}	MAE	RMSE	SMAPE [%]
1 hour	42	23.23	0.9350	2.3549	49.88 %
6 hours	34	16.09	1.9897	4.2573	80.65 %
12 hours	20	8.71	2.4123	4.9315	91.82 %
1 day	18	6.00	2.7811	5.3624	102.09 %
2 days	12	4.54	3.0248	5.7750	106.60 %
3 days	10	3.46	3.0045	5.8903	108.01 %
5 days	7	2.08	3.2856	6.2383	112.15 %
7 days	6	1.75	3.5347	6.6072	118.18 %

Table 4.21: Detailed precision and complexity results of TinyRBF network application for DHT11 sensor calibration, where N_{max} and N_{avg} are respectively the max and mean number of nodes during the learning process.

of 21.3% in measurement error.

This analysis demonstrates the robustness of the TinyRBF network in reducing sensor error across varying conditions of reference availability. While the naive model performs adequately for shorter t_{ref}, the TinyRBF network maintains a strong advantage, particularly for longer and moderate t_{ref} intervals, showcasing its effectiveness in adapting to sensor drift and improving overall accuracy.

4.4 General Purpose Applications

This section presents the results of applying the on-device learning approach proposed in Chapter 3 in contexts beyond precision agriculture, yet still closely tied to the challenge of sensor calibration. Specifically, Section 4.4.1 demonstrates the use of the TinyRBF network for calibrating a non-invasive glucose sensor, while Section 4.4.2 introduces the initial calibration results of Inertial Measurement Unit measurements, comparing their precision and complexity with traditional backpropagation-based methods.

4.4.1 Non-Invasive Glucose Sensor

Diabetes poses a major global health challenge, requiring precise blood glucose monitoring for effective management. Continuous Glucose Monitoring (CGM) devices provide a minimally invasive solution but depend on accurate calibration models to enhance reliability.

Various neural network architectures were explored to predict time errors in CGM sensor readings in [63], focusing on achieving high accuracy with low computational complexity. Using simulated data, models were trained and evaluated, with the Legendre Memory Unit (LMU) and Temporal Convolutional Network (TCN) emerging as promising candidates. These architectures reduced sensor reading errors to 24.22 mg/dL and 25.34 mg/dL, representing reductions of 40.6% and 37.9%, respectively. Additionally, the study examined the effect of sensor calibration frequency on prediction accuracy, finding optimal results with calibrations every three days, achieving reading errors of 14.42 mg/dL and 15.18 mg/dL. The TinyRBF approach was employed to facilitate efficient and adaptive on-device learning for personalized glucose prediction, overcoming the limitations of sequential data processing inherent in LMU and TCN architectures.

The database used in [63] and in this experiment was created through simulation rather than real patient data, primarily due to feasibility concerns. It was generated using a Python implementation of the FDA-approved UVA/Padova Simulator [64], modeling ten virtual adult patients. The database consists of glucose responses over

a 15-day period, with measurements taken every 3 minutes to match the real sensor's sampling frequency. Daily meal plans, or "scenarios" were designed to reflect realistic dietary patterns, including meal timings, probabilities, and nutritional values. The sensor response model incorporated factors such as blood glucose-to-interstitial glucose kinetics, sensor measurement errors, white noise, and sensor drift due to degradation over time. This multifaceted simulation approach ensured the database accurately reflected real-world sensor behaviors.

Given that sensor data is collected at consistent intervals of 3 minutes, the network alternates between performing inference and training at each sampling step, as outlined in Sections 3.3 and 3.4. In this experiment, calibration is conducted at the first sample and then periodically with a calibration interval of t_{ref}. The values for t_{ref} are 1 hour, 6 hours, 12 hours, 1 day, 2 days, 3 days, 5 days, and 7 days. The goal is to evaluate the robustness of the calibration method across different calibration frequencies.

In order to identify the best configuration for the TinyRBF network evolution, a grid search approach was applied. The results of the grid search, including the best hyperparameters found, are summarized in Table 4.22.

t_{ref}	r_{init}	ε	β	η	γ	M	α	W
6 hours	125.0	5.0	0.75	0.2	1.2	72	0.01	36
12 hours	125.0	5.0	0.75	0.3	1.4	72	0.01	6
1 day	125.0	5.0	0.75	0.4	1.8	72	0.01	1
2 days	125.0	5.0	0.75	0.5	2.0	72	0.01	1
3 days	125.0	5.0	0.75	0.5	1.8	72	0.01	1
5 days	125.0	5.0	0.75	0.6	2.0	72	0.01	1
7 days	125.0	5.0	0.75	0.6	1.8	72	0.01	1

Table 4.22: Best hyperparameters that govern the evolution of the TinyRBF network for each different recalibration frequency.

Table 4.23 presents a comprehensive summary of the network's performance and structure across different recalibration periods. It includes the maximum and mean

number of neurons used during training and inference and precision metrics, highlighting the network's adaptability and resource efficiency. The complexity is extremely reduced, since the maximum number of RBF nodes allocated in all configurations is 12. This value corresponds to 49 parameters and 196 bytes of memory used, calculated using the Equation 3.21. When comparing measurement errors for $t_{\text{ref}} = 3$ days, the TinyRBF network outperforms the LMU and TCN approaches, which achieve MAE values of 14.42 mg/dL and 15.18 mg/dL, respectively. The TinyRBF network achieves a MAE of 12.10 mg/dL, representing reductions of 16.1% and 20.3%, respectively. From a complexity standpoint, the LMU and TCN networks require 296 and 1770 parameters, respectively. In contrast, the on-device learning approach averages just 1.02 neurons, equivalent to approximately 5 parameters, demonstrating significantly lower complexity.

t_{ref}	Max Neurons	Mean Neurons	MAE [mg/dL]	RMSE [mg/dL]	SMAPE [%]
6 hours	12	3.20	8.5103	11.0721	35.3965
12 hours	8	1.90	9.4053	12.0706	38.3107
1 day	2	1.05	10.2942	13.0884	40.2267
2 days	3	1.01	11.3043	14.2609	44.4386
3 days	2	1.02	12.0996	15.1670	48.2219
5 days	2	1.00	14.1254	17.5359	56.9148
7 days	2	1.00	16.4767	20.2713	67.6451

Table 4.23: Complexity and precision results for each recalibration frequency.

The reduction in glucose sensor measurement error compared to the baseline dataset ranges from 87.8%, when the reference value is every 6 hours, to 76.5%, when the calibration occurs with a period of 7 days, as shown in Figure 4.29.

Figure 4.30 shows the MAE and RMSE values after calibration using the TinyRBF network, with the x-axis adjusted to represent a continuous scale. The graph highlights that error growth is significantly sublinear for calibration intervals shorter than one day, while it becomes nearly linear for intervals with fewer than one calibration

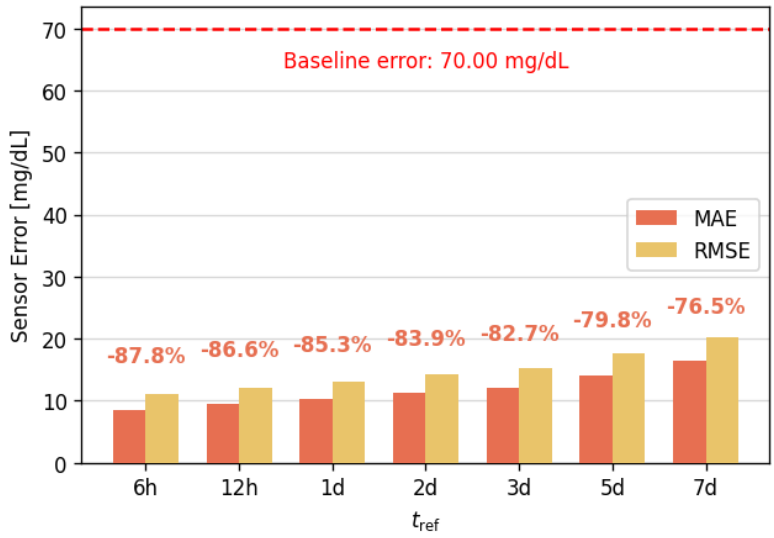


Figure 4.29: Mean sensor precision after on-device error compensation for each recalibration frequency.

per day.

A C implementation of the proposed solution was specifically developed to target the *Intelligent Sensor Processing Unit* (ISPU) device[12] by STMicroelectronics, for validation purposes. The ISPU is an Inertial Measurement Unit (IMU) sensor that embeds an ultralow power, computationally efficient, high-performance programmable core that can perform signal processing and execute AI algorithms at the edge. This device can process data from internal sensors (accelerometer, gyroscope, and temperature), or from an external sensors hub, or directly from a master microcontroller, which provides it with all the data it needs. The ISPU can run at 10 or 20 MHz and is equipped with 32 KB of program RAM, 8 KB of data RAM and an FPU supporting addition, subtraction and multiplication operations. Any custom program can be loaded in the core using a proprietary toolchain that allows developing in C code.

The implementation was carefully designed to ensure compatibility with the resource-

[12] `https://www.st.com/content/st_com/en/campaigns/ispu-ai-in-sensors.html`

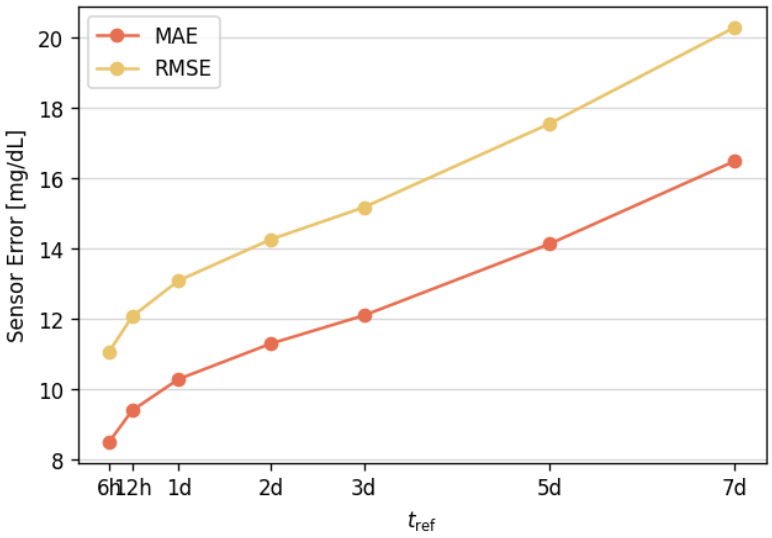


Figure 4.30: Mean sensor precision after on-device error compensation for each recalibration frequency with the x-axis adjusted to reflect a continuous scale.

constrained ISPU platform while adhering to strict limitations, such as avoiding external libraries and relying solely on static memory allocation. This limitation necessitates setting a maximum number of neurons that the hidden layer of the network can contain, as all the memory required by the model must be allocated at the beginning. The implementation of the TinyRBF learning algorithm, as described in Section 3.4, has been slightly adjusted to include a check that verifies when the maximum complexity is reached, which is reflected in the allocation condition outlined in Equation 3.12.

Table 4.24 reports experimental results of the deployment of the network on the ISPU device using a 32-bit floating point representation for network parameters. This table presents the program RAM usage, which is determined by the code, and the data RAM footprint, which is based on the model parameters. It also includes the average inference and training times. Specifically, t_{inf} represents the average inference time in milliseconds, while t_{learn} measures the average time required to perform a model update. However, the first calibration step has minimal complexity, as the empty net-

work can only add the first RBF node, significantly skewing the average. To address this, the parameter t'_{learn} is introduced, which excludes the first training step, providing a more accurate measure of the learning time.

t_{ref}	Max Neurons	Program RAM [bytes]	Data RAM [bytes]	t_{inf} [ms]	t_{learn} [ms]	t'_{learn} [ms]
6 hours	15	17,072	384	8.68	9.30	9.45
12 hours	10	17,072	304	6.16	6.22	6.43
1 day	5	17,072	224	2.71	3.00	3.20
2 days	5	17,072	224	2.72	2.86	3.25
3 days	5	17,072	224	2.71	2.59	3.20
5 days	5	17,072	224	2.71	2.18	3.20
7 days	5	17,072	224	2.71	2.18	3.19

Table 4.24: Experimental results of the deployment on the ISPU device of the proposed on-device learning solution using 32-bit floating point representation for network parameters.

The 16-bit quantization implementation was introduced to optimize memory usage and computational efficiency while maintaining model performance. By reducing model parameters' precision from 32-bit to 16-bit, the implementation ensures a significant reduction in both memory requirements and processing time, making it more suitable for deployment on resource-constrained devices. Table 4.25 presents the experimental results of this 16-bit quantized implementation in terms of memory footprint and inference and learning time. On average, the inference time decreased by 33.5%, while the reduction in training time was less pronounced. The program memory slightly increased due to the added complexity of managing the quantized parameters, whereas the data RAM usage was reduced by approximately 15%. Overall, the reduction in data memory and processing times became more pronounced as the network complexity increased.

t_{ref}	Max Neurons	Program RAM [bytes]	Data RAM [bytes]	t_{inf} [ms]	t_{learn} [ms]	t'_{learn} [ms]
6 hours	15	18,220	296	4.65	5.90	5.97
12 hours	10	18,220	244	3.34	4.32	4.40
1 day	5	18,220	196	2.07	2.78	2.90
2 days	5	18,220	196	1.94	2.49	2.73
3 days	5	18,220	196	1.93	2.35	2.73
5 days	5	18,220	196	1.93	2.08	2.71
7 days	5	18,220	196	1.93	1.95	2.52

Table 4.25: Experimental results of the deployment on the ISPU device of the proposed on-device learning solution using 16-bit quantization for network parameters.

4.4.2 Inertial Measurement Unit Sensor

This experiment addresses the challenge of compensating for time-varying Inertial Measurement Unit (IMU) readings under unpredictable stresses throughout the sensor's lifespan. The proposed solution must have low computational complexity suitable for a 10 MHz digital signal processor and a memory footprint within 40 KiB of embedded RAM. It must provide accurate error estimation using the sensor's computational resources and support real-time, online processing. Key considerations for the solution design include periodic recalibration to counteract sensor aging, robust handling of both linear and nonlinear data corruptions caused by calibration loss, and coherent data production to adapt to previously unknown measurement errors. The solution must avoid computationally intensive back-propagation-based learning methods, operate effectively on resource-constrained devices, and perform streaming inferences for real-time compensation. The goal is a compact, efficient implementation that fulfills these requirements while remaining deployable on tiny embedded systems.

The proposed system architecture compensates for MEMS sensor inaccuracies by periodically evaluating sensor outputs against a high-precision Golden Reference (GR), which provides ground truth values. A compensation block integrates a model

trained using hyperparameters aligned with the GR. The system applies predefined stimuli to the MEMS sensor, feeding the outputs into the model for inference. The compensation error is calculated as the element-wise difference between the MEMS and GR responses to the same stimuli. This error is added to the MEMS output, producing a compensated output that is compared against the GR values. To maintain accuracy, the model is periodically re-trained using the same predefined stimuli.

The topology of the TinyRBF network consists of an input layer, hidden layers, and an output layer. In this experiment, the input to the network is the uncompensated acceleration data from the MEMS sensor. The input array is built by concatenating the current values along with the previous four time steps for each of the three axes (X, Y, and Z), resulting in an array of 15 elements (5 values per axis). The network uses three output neurons, one for each axis (X, Y, and Z), to compute the compensation errors. The output layer is designed to produce the compensation values for each axis, which are added to the raw MEMS data to generate the compensated sensor readings.

Three different datasets were employed to validate the on-device learning calibration approach.

Synthetic MEMS Accelerometer Dataset (SMAD)

Due to the lack of real sensor data under both movement and aging conditions, a synthetic accelerometer dataset was generated. The Golden Reference (GR) sensor, modeled as an identity function, provided ground truth values. Both the GR and MEMS models were exposed to identical stimuli (impulse, square, and sine waves), generating 5 minutes of data.

The MEMS model included a linear filter to simulate sensor errors, Gaussian noise, and time-varying bias. The output was then processed through a nonlinear Volterra filter. Test data were synthesized using sine, square, and sinc functions with random parameters, ensuring variability. The dataset, scaled between -4g and 4g, was used for neural network training and simulated complex real-world scenarios.

Synthetic MEMS Accelerometer Dataset with Recoverable Bias (SMAD-RB)

The updated dataset generation model addresses the issue of recovering the bias

trend in the data. In this version, linear corruption is applied without adding the bias to the input signal. A new error matrix amplifies the cross-sensitivity of the data. The second-order Volterra filter is then applied, followed by the addition of Gaussian noise and time-varying bias. The output is scaled between -4g and 4g.

This model allows the bias trend to be recovered, unlike the previous model where nonlinear corruption made it irrecoverable. The bias drift after one day was quantified. Test data were generated over four days, with the bias initialized based on its value at the start of each day, allowing the dataset to simulate the MEMS aging process for network training.

Device Specific MEMS Accelerometer Dataset - Thermal Stress (DSMAD-TS)

To validate the proposed methodology, data were collected by deploying an IMU in the field under thermal stress, a major source of measurement error. The LSM6DSV 6-axis sensor, which includes a 3-axis accelerometer, was used, with data gathered only from the accelerometer. Twenty-one devices under test (DUT) were evaluated, placed on PCBs to ensure that gravitational acceleration was applied only to the Z axis. Thermal stress was applied covering temperatures from -40°C to 85°C. Data were collected at 15 Hz for 3 hours and 15 minutes per DUT. The Golden Reference (GR) was assumed to measure 0g on the X and Y axes, and 1g on the Z axis. The first 5 minutes of data were used for network initialization and training. Recalibrations were performed at 2340 seconds (when the temperature reached 93.30°C) and 6599.07 seconds (when the temperature dropped to -40.46°C), as these times marked the most significant environmental changes. Inferences were made before and after these recalibrations to evaluate the methodology's performance.

The performance of the approach proposed in this study is presented in Tables 4.26, 4.27, and 4.28. These results are compared with four methods that use backpropagation, as described in [65]. The first two models use 32-bit floating-point precision, while the third and fourth models are quantized versions of the first two, employing a 16-bit quantization-aware training technique. The second and fourth models

feature three parallel branches, which are concatenated before the output layer. The last row of the tables presents the the TinyRBF network using 32-bit floating-point data representation.

The performance of the models was evaluated using two metrics: *Mean Compensation Value* (MCV) and *Mean Compensation Percentage* (MCP).

MCV is defined by Equation where i is the axis (X, Y, Z), $MEMS_i$ is the sensor output, GR_i is the Golden Reference, and $Comp_i$ is the compensated output.

$$\text{MCV} = \frac{1}{3}\sum_{i=1}^{3}(\text{MAE}(MEMS_i, GR_i) - \text{MAE}(Comp_i, GR_i)) \tag{4.6}$$

It is calculated as the average of the difference between the MAE of the MEMS output and the MAE of the compensated output, across all three axes (X, Y, Z). Positive MCV values indicate an improvement in compensation, while negative values suggest a deterioration in accuracy.

MCP represents the improvement in terms of error compensation percentage. It is calculated by dividing the MCV by the initial MAE and multiplying by 100:

$$\text{MCP} = \frac{\text{MCV}}{\text{MAE}(MEMS_{\text{i}}, GR_{\text{i}})} \times 100 \tag{4.7}$$

This percentage indicates the proportion of the error that has been compensated by the model.

Model	**MACC (per sample)**	**Trainable parameters**	**Average MCV [mg]**	**Average MCP**
Conv1D-Dense	888	903	4.70	44.97%
Parallel Conv1D	649	651	6.30	59.30%
Quantized Conv1D-dense	868	903	4.67	44.58%
Quantized Parallel Conv1D	613	651	4.79	45.46%
RBF-DSMAD-TS	843	196	9.36	83.30%

Table 4.26: Performance comparison between the BP-based methods and the devised TinyRBF network on IMU error compensation using the DSMAD-TS dataset.

Three different TinyRBF Networks were evaluated in this experiment:

Model	MACC (per sample)	Trainable parameters	Average MCV [g]	Average MCP
Conv1D-Dense	888	903	1.91	53.93%
Parallel Conv1D	649	651	2.10	60.77%
Quantized Conv1D-dense	868	903	1.66	46.81%
Quantized Parallel Conv1D	613	651	1.97	55.45%
RBF-SMAD	1081	250	1.62	45.72%

Table 4.27: Performance comparison between the BP-based methods and the presented TinyRBF network on IMU error compensation using the SMAD dataset.

Model	MACC (per sample)	Trainable parameters	Average MCV [g]	Average MCP
Conv1D-Dense	888	903	1.55	43.99%
Parallel Conv1D	649	651	1.63	46.24%
Quantized Conv1D-dense	868	903	2.46	69.66%
Quantized Parallel Conv1D	613	651	2.45	69.26%
RBF-SMAD-RB	135	34	2.07	58.47%

Table 4.28: Performance comparison between the BP-based methods and the presented TinyRBF network on IMU error compensation using the SMAD-RB dataset.

RBF-SMAD Trained on the SMAD dataset, this network used 13 RBF units (1,000 bytes memory footprint) and 1,081 MACC. With 250 trainable parameters, it performed similarly to other models, achieving an average MCV of 1.62 g and an MCP of 45.72%. The dataset was challenging due to a non-linear bias trend and issues simulating IMU aging, making training difficult.

RBF-SMAD-RB Trained on the SMAD-RB dataset (with recoverable bias), this network required only 1.6 RBF units (136 bytes memory footprint) and 135 MACC. With just 34 parameters, it achieved an average MCV of 2.07 and an MCP of 58.47%, outperforming floating-point networks but falling short of the quantized ones.

RBF-DSMAD-TS Trained on the DSMAD-TS dataset, this network required an average of 10 RBF units (784 bytes memory footprint) and 843 MACC. It had 196 trainable parameters, fewer than other models (651–903). This network achieved the best performance, with an average MCV of 9.36 mg and an MCP of 83.30%, indicating superior compensation performance.

In summary, the RBF-DSMAD-TS network offered the best performance, while the RBF-SMAD-RB network was the most efficient in terms of memory and computation.

Conclusion

This Thesis adopted a use case-driven structure to address practical challenges and develop generalizable and innovative solutions for advanced edge sensing in a variety of applications, including precision agriculture. By starting from real-world problems, the research aimed to bridge the gap between theoretical advancements and their applicability in resource-constrained environments. The core contributions of this work lie in proposing and validating a novel approach for on-device learning and demonstrating its effectiveness across diverse applications.

The proposed solution represents a significant advancement in the field of on-device machine learning, particularly under severe resource constraints. The approach enables online learning with streaming data from sensors, addressing key challenges such as concept drift and the need for real-time adaptability. The solution achieves competitive accuracy while maintaining an exceptionally small memory footprint and low computational complexity for both inference and learning. These characteristics make it ideal for deployment on tiny devices such as microcontrollers and IoT-enabled sensors.

One of the most important outcomes of this work is the successful implementation and validation of real-time, in-field correction of environmental data. This capability was demonstrated under complex operating conditions during a long-term field deployment. The method proved robust in handling dynamic environmental stresses and long-term drift, delivering consistent and reliable data over extended periods.

This Thesis also compared the TinyRBF network to a naive baseline model, which predicts the last received reference value, to evaluate whether the added com-

plexity of the on-device learning approach is justified. The results showed that while the naive model performs adequately with frequent reference updates, its accuracy declines significantly with longer intervals. In contrast, the TinyRBF network consistently outperformed the naive model across all intervals, demonstrating superior adaptability to sensor drift. Additionally, the naive model occasionally worsened measurement accuracy, whereas the TinyRBF network consistently reduced errors, even if its impact varied. These findings highlight the effectiveness and necessity of the TinyRBF network for robust sensor error compensation.

The effectiveness of the proposed approach extends beyond environmental monitoring. Results from its application to contexts and sensors other than precision agriculture further underline the versatility and generalizability of the solution. By enabling robust learning and adaptation in low-cost, resource-limited devices, the method opens new possibilities for deploying advanced machine learning models in a wide range of domains, from smart wearable sensors to industrial IoT and beyond.

Based on the findings and contributions of this Thesis, several directions for future research can be identified.

Although the proposed TinyRBF approach has been validated extensively on environmental sensors and preliminarily on other sensors, such as glucose and IMU, its application could be extended to other domains, such as industrial IoT, healthcare monitoring, and wearables. Investigating the adaptability of the method to different sensor modalities and use cases would further demonstrate its generalizability and scalability.

The proposed solution achieves low computational complexity, however, further research could focus on improving energy efficiency, especially for battery-powered devices. This could involve integrating energy-aware algorithms or exploiting ultra-low-power hardware.

Expanding the duration and scope of field deployments would provide deeper insights into the model's performance over time, especially its ability to handle changing environmental and operational conditions. This could include deployments in more challenging environments or over multiple seasons to assess long-term robustness.

Additionally, future research could investigate methods to automate model adaptation, such as self-optimizing algorithms that dynamically adjust hyperparameters or architecture based on evolving data distributions and resource constraints.

Exploring collaborative or federated learning techniques over a network of resource-constrained devices could enable the solution to leverage distributed data while preserving privacy and minimizing communication overhead. This would be particularly valuable in large-scale deployments, such as smart agriculture or environmental monitoring networks.

As edge AI continues to evolve, future studies could benchmark the proposed solution against emerging learning algorithms and hardware optimizations. This would help refine the approach and ensure it remains competitive in a rapidly evolving field.

By addressing these areas, future research can build on the fundamental contributions of this Thesis, further improving the capabilities and applicability of on-device learning in resource-constrained environments.

In summary, this Thesis contributes to the growing number of researches focused on edge AI by presenting a scalable, efficient, and practical solution for on-device learning and sensor calibration. Its findings pave the way for future innovations in deploying intelligent systems in resource-constrained environments, where traditional machine learning methods have often struggled to succeed.

Bibliography

[1] António Monteiro, Sérgio Santos, and Pedro Gonçalves. Precision agriculture for crop and livestock farming—brief review. *Animals*, 11(8), 2021.

[2] Francis J Pierce and Peter Nowak. Aspects of precision agriculture. *Advances in agronomy*, 67:1–85, 1999.

[3] Abhinav Sharma, Arpit Jain, Prateek Gupta, and Vinay Chowdary. Machine learning applications for precision agriculture: A comprehensive review. *IEEE Access*, 9:4843–4873, 2020.

[4] Rajendra P Sishodia, Ram L Ray, and Sudhir K Singh. Applications of remote sensing in precision agriculture: A review. *Remote sensing*, 12(19):3136, 2020.

[5] Giorgia Bucci, Deborah Bentivoglio, Adele Finco, et al. Precision agriculture as a driver for sustainable farming systems: state of art in literature and research. *Calitatea*, 19(S1):114–121, 2018.

[6] Jorge A Delgado, Nicholas M Short Jr, Daniel P Roberts, and Bruce Vandenberg. Big data analysis for sustainable agriculture on a geospatial cloud framework. *Frontiers in Sustainable Food Systems*, 3:54, 2019.

[7] David J Mulla. Twenty five years of remote sensing in precision agriculture: Key advances and remaining knowledge gaps. *Biosystems engineering*, 114(4):358–371, 2013.

[8] Abhishek Khanna and Sanmeet Kaur. Evolution of internet of things (iot) and its significant impact in the field of precision agriculture. *Computers and electronics in agriculture*, 157:218–231, 2019.

[9] Showkat Ahmad Bhat and Nen-Fu Huang. Big data and ai revolution in precision agriculture: Survey and challenges. *Ieee Access*, 9:110209–110222, 2021.

[10] Reza Rahmadian and Mahendra Widyartono. Autonomous robotic in agriculture: a review. In *2020 third international conference on vocational education and electrical engineering (ICVEE)*, pages 1–6. IEEE, 2020.

[11] Elias Fereres and María Auxiliadora Soriano. Deficit irrigation for reducing agricultural water use. *Journal of experimental botany*, 58(2):147–159, 2007.

[12] Jonas Franke and Gunter Menz. Multi-temporal wheat disease detection by multi-spectral remote sensing. *Precision Agriculture*, 8:161–172, 2007.

[13] Danco Davcev, Kosta Mitreski, Stefan Trajkovic, Viktor Nikolovski, and Nikola Koteli. Iot agriculture system based on lorawan. In *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 1–4. IEEE, 2018.

[14] Mohammed Jouhari, Nasir Saeed, Mohamed-Slim Alouini, and El Mehdi Amhoud. A Survey on Scalable LoRaWAN for Massive IoT: Recent Advances, Potentials, and Challenges. *IEEE Communications Surveys & Tutorials*, 25(3):1841–1876, 2023.

[15] Brian G Leib, Jay D Jabro, and Gary R Matthews. Field evaluation and performance comparison of soil moisture sensors. *Soil Science*, 168(6):396–408, 2003.

[16] Limin Yu, Wanlin Gao, Redmond R Shamshiri, Sha Tao, Yanzhao Ren, Yanjun Zhang, and Guilian Su. Review of research progress on soil moisture sensor technology. *International Journal of Agricultural and Biological Engineering*, 14, 2021.

[17] Ala Saleh Alluhaidan, Rab Nawaz Bashir, Rashid Jahangir, Radwa Marzouk, Oumaima Saidani, and Roobaea Alroobaea. Machine learning and fog computing-enabled sensor drift management in precision agriculture. *IEEE Sensors Journal*, 24(22):36953–36970, 2024.

[18] Deboleena Roy, Gopalakrishnan Srinivasan, Priyadarshini Panda, Richard Tomsett, Nirmit Desai, Raghu Ganti, and Kaushik Roy. Neural Networks at the Edge. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 45–50, 2019.

[19] Yongho Kim, Seongha Park, Sean Shahkarami, Rajesh Sankaran, Nicola Ferrier, and Pete Beckman. Goal-driven scheduling model in edge computing for smart city applications. *Journal of Parallel and Distributed Computing*, 167:97–108, 2022.

[20] J. Aleotti, M. Amoretti, A. Nicoli, and S. Caselli. A Smart Precision-Agriculture Platform for Linear Irrigation Systems. In *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6, 2018.

[21] M. Amoretti, D. L. Rizzini, G. Penzotti, and S. Caselli. A Scalable Distributed System for Precision Irrigation. In *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2020.

[22] Bam Bahadur Sinha and R. Dhanalakshmi. Recent advancements and challenges of Internet of Things in smart agriculture: A survey. *Future Generation Computer Systems*, 126:169–184, 2022.

[23] Sameer Qazi, Bilal A. Khawaja, and Qazi Umar Farooq. IoT-Equipped and AI-Enabled Next Generation Smart Agriculture: A Critical Review, Current Challenges and Future Trends. *IEEE Access*, pages 21219–21235, 2022.

[24] Davide Brunelli, Andrea Albanese, Donato d'Acunto, and Matteo Nardello. Energy neutral machine learning based iot device for pest detection in precision agriculture. *IEEE Internet of Things Magazine*, 2(4):10–13, 2019.

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

[26] Nicola Coppedè, Michela Janni, Manuele Bettelli, Calogero Leandro Maida, Francesco Gentile, Marco Villani, Roberta Ruotolo, Salvatore Iannotta, Nelson Marmiroli, Marta Marmiroli, and Andrea Zappettini. An in Vivo Biosensing, Biomimetic Electrochemical Transistor with Applications in Plant Science and Precision Farming. *Scientific Reports*, 7(1):16195, 2017.

[27] Juan Pablo Giraldo, Honghong Wu, Gregory Michael Newkirk, and Sebastian Kruss. Nanobiotechnology Approaches for Engineering Smart Plant Sensors. *Nature Nanotechnology*, 14(6):541–553, 2019.

[28] Francesco Gentile, Filippo Vurro, Michela Janni, Riccardo Manfredi, Francesco Cellini, Angelo Petrozza, Andrea Zappettini, and Nicola Coppedè. A Biomimetic, Biocompatible OECT Sensor for the Real-Time Measurement of Concentration and Saturation of Ions in Plant Sap. *Advanced Electronic Materials*, page 2200092, 2022.

[29] Antonio Ruiz-Gonzalez, Harriet Kempson, and Jim Haseloff. In vivo sensing of ph in tomato plants using a low-cost and open-source device for precision agriculture. *Biosensors*, 12(7), 2022.

[30] D. Bernards and G. Malliaras. Steady-State and Transient Behavior of Organic Electrochemical Transistors. *Advanced Functional Materials*, 17(17):3538–3544, 2007.

[31] Francesco Gentile, Davide Delmonte, Massimo Solzi, Marco Villani, Salvatore Iannotta, Andrea Zappettini, and Nicola Coppedé. A theoretical model for the time varying current in organic electrochemical transistors in a dynamic regime. *Organic Electronics*, 35:59–64, 2016.

[32] Francesco Gentile, Filippo Vurro, Francesco Picelli, Manuele Bettelli, Andrea Zappettini, and Nicola Coppedè. A Mathematical Model of OECTs with Variable Internal Geometry. *Sensors and Actuators A: Physical*, 304:111894, 2020.

[33] Firas Bayram, Bestoun S. Ahmed, and Andreas Kassler. From concept drift to model degradation: An overview on performance-aware drift detectors, 2022.

[34] David E Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation: The basic theory. In *Backpropagation*. Psychology Press, 2013.

[35] Alexander Gepperth and Barbara Hammer. Incremental learning algorithms and applications. In *European Symposium on Artificial Neural Networks (ESANN)*, Bruges, Belgium, 2016.

[36] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 2017.

[37] Steve Klabnik and Carol Nichols. *The Rust Programming Language (Covers Rust 2018)*. No Starch Press, 2019.

[38] Tunç Uzlu and Ediz Şaykol. On utilizing rust programming language for internet of things. In *2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 93–96. IEEE, 2017.

[39] Sauptik Dhar, Junyao Guo, Jiayi (Jason) Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. A Survey of On-Device Machine Learning: An Algorithms and Learning Theory Perspective. *ACM Transactions on Internet of Things*, 2(3):1–49, August 2021.

[40] Visal Rajapakse, Ishan Karunanayake, and Nadeem Ahmed. Intelligence at the Extreme Edge: A Survey on Reformable TinyML. *arXiv:2204.00827 [cs, eess]*, April 2022. arXiv: 2204.00827.

[41] Shuai Zhu, Thiemo Voigt, JeongGil Ko, and Fatemeh Rahimian. On-device Training: A First Overview on Existing Systems, December 2022. arXiv:2212.00824 [cs].

[42] Geoffrey Hinton. The Forward-Forward Algorithm: Some Preliminary Investigations, December 2022. arXiv:2212.13345 [cs].

[43] Alexander Ororbia and Ankur Mali. The Predictive Forward-Forward Algorithm, April 2023. arXiv:2301.01452 [cs].

[44] Fabrizio De Vita, Rawan M. A. Nawaiseh, Dario Bruneo, Valeria Tomaselli, Marco Lattuada, and Mirko Falchetto. μ-FF: On-Device Forward-Forward Training Algorithm for Microcontrollers. In *SMARTCOMP 2023*. IEEE, June 2023.

[45] Giorgia Dellaferrera and Gabriel Kreiman. Error-driven Input Modulation: Solving the Credit Assignment Problem without a Backward Pass. In *Proceedings of the 39th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, July 2022.

[46] Ravi Francesco Srinivasan, Francesca Mignacco, Martino Sorbaro, Maria Refinetti, Avi Cooper, Gabriel Kreiman, and Giorgia Dellaferrera. Forward Learning with Top-Down Feedback: Empirical and Analytical Characterization, 2023.

[47] Danilo Pietro Pau and Fabrizio Maria Aymone. Suitability of Forward-Forward and PEPITA Learning to MLCommons-Tiny benchmarks. In *COINS 2023*, pages 1–6, Berlin, Germany, July 2023. IEEE.

[48] Adam Kohan, Edward A. Rietman, and Hava T. Siegelmann. Signal Propagation: The Framework for Learning and Inference in a Forward Pass. *IEEE Trans. on Neural Networks and Learning Systems*, January 2023.

[49] M.J. Hudak. RCE networks: an experimental investigation. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume i, pages 849–854, Seattle, WA, USA, 1991. IEEE.

[50] Danilo Pietro Pau, Andrea Pisani, Fabrizio Maria Aymone, and Gianluigi Ferrari. TinyRCE: Multipurpose Forward Learning for Resource Restricted Devices. *IEEE Sensors Letters*, 7(10):1–4, October 2023.

[51] Yue Wu, Hui Wang, Biaobiao Zhang, and K.-L. Du. Using Radial Basis Function Networks for Function Approximation and Classification. *ISRN Applied Mathematics*, 2012:1–34, March 2012.

[52] John Platt. A Resource-Allocating Network for Function Interpolation. *Neural Computation*, 3(2):213–225, June 1991.

[53] Lu Yingwei, N. Sundararajan, and P. Saratchandran. A Sequential Learning Scheme for Function Approximation Using Minimal Radial Basis Function Neural Networks. *Neural Computation*, February 1997.

[54] G.-B. Huang, P. Saratchandran, and N. Sundararajan. An Efficient Sequential Learning Algorithm for Growing and Pruning RBF (GAP-RBF) Networks. *IEEE Trans. on Systems, Man and Cybernetics, Part B (Cybernetics)*, December 2004.

[55] Guang-Bin Huang, P. Saratchandran, and N. Sundararajan. A generalized growing and pruning rbf (ggap-rbf) neural network for function approximation. *IEEE Transactions on Neural Networks*, 16(1):57–67, 2005.

[56] Noriaki Kouda and Nobuyuki Matsui. On the function approximation in Restricted Coulomb Energy neural network with Gaussian Radial Basis Function. In *2010 World Automation Congress*, pages 1–5, December 2010.

[57] Lijie Jia, Wenjing Li, and Junfei Qiao. An online adjusting rbf neural network for nonlinear system modeling. *Applied Intelligence*, 53(1):440–453, 2023.

[58] Alexei Botchkarev. Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology. *arXiv preprint arXiv:1809.03006*, 2018.

[59] Francesco Saccani, Manuele Bettelli, Francesco Gentile, and Michele Amoretti. Energy-efficient OECT Sensor Data Analysis on Constrained Edge Devices . In *2023 IEEE International Conference on Cloud Engineering (IC2E)*, pages 51–58, Los Alamitos, CA, USA, September 2023. IEEE Computer Society.

[60] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[61] Francesco Saccani, Danilo Pau, and Michele Amoretti. In-sensor learning for pressure self-calibration. In *2024 IEEE Sensors Applications Symposium (SAS)*, pages 1–6, 2024.

[62] Francesco Saccani, Danilo Pau, and Michele Amoretti. Learning pressure sensor drifts from zero deployability budget. *IEEE Sensors Letters*, 8(8):1–4, 2024.

[63] Costanza Cenerini, Anna Sabatini, Luca Vollero, and Danilo Pau. Optimizing glucose sensor calibration with lightweight neural networks: A comparative study. *IEEE Sensors Letters*, 8(9):1–4, 2024.

[64] Chiara Dalla Man, Francesco Micheletto, Dayu Lv, Marc Breton, Boris Kovatchev, and Claudio Cobelli. The uva/padova type 1 diabetes simulator: new features. *Journal of diabetes science and technology*, 8(1):26–34, 2014.

[65] Matteo Cardoni, Danilo Pietro Pau, Kiarash Rezaei, and Camilla Mura. Inertial measurement unit self-calibration by quantization-aware and memory-parsimonious neural networks. *Electronics*, 13(21), 2024.

Acknowledgements

La borsa di dottorato è stata cofinanziata con risorse del Programma Operativo Nazionale Ricerca e Innovazione 2014-2020, risorse FSE REACT-EU Azione IV.4 “Dottorati e contratti di ricerca su tematiche dell’innovazione” e Azione IV.5 “Dottorati su tematiche Green”.