Designing Tiny Ultra-Low-Power Neural Network Hardware Accelerators for In-Sensor Computing

Paola Vitolo

UNIVERSITY OF SALERNO

DEPARTMENT OF INDUSTRIAL ENGINEERING

Ph.D. Course in Industrial Engineering
Curriculum in Electronic Engineering - XXXVII Cycle

DESIGNING TINY ULTRA-LOW-POWER NEURAL NETWORK HARDWARE ACCELERATORS FOR IN-SENSOR COMPUTING

Supervisors
Prof. Alfredo Rubino
Prof. Gian Domenico Licciardo

Ph.D. Student
Paola Vitolo

Scientific Referees
Prof. Maurizio Valle
Prof. Rajkumar Kubendran

Ph.D. Course Coordinator
Prof. Massimo De Santo

Acknowledgments

I am deeply grateful to all the people who supported me throughout my Ph.D. journey and contributed to my growth both as a researcher and as a person.

First and foremost, I would like to thank my supervisors, Prof. Gian Domenico Licciardo and Prof. Alfredo Rubino, for their constant guidance, trust, and encouragement, and for introducing me to both academic and industrial research with integrity and enthusiasm.

I would like to warmly thank Prof. Andreou for supervising my research period at Johns Hopkins University, for believing in me, and for offering me one of the most rewarding research experiences of my Ph.D.

My gratitude goes to my industrial supervisor at STMicroelectronics, Danilo Pau, for his guidance, enthusiasm, and invaluable advice on both research and professional development. I am also grateful to all the colleagues at STMicroelectronics and to the professionals I had the opportunity to work with during my industrial and collaborative research activities.

I would like to thank my colleagues and collaborators at the laboratory, as well as my Ph.D. peers and thesis students, for the stimulating discussions, teamwork, and shared experiences that greatly enriched my doctoral journey.

I am grateful to the colleagues and friends I met during my research period abroad, who made me feel welcome and supported, and to all the researchers and professionals I encountered at conferences, workshops, and collaborative projects.

Finally, I would like to thank my family, friends, and loved ones for their constant presence, patience, and unconditional support during this long and demanding journey.

This achievement would not have been possible without them and is not mine alone. It belongs to all those who have shared this path with me, in big ways and small, and whose kindness, wisdom, and friendship have shaped both my research and my life.

“La curiosità è l’anima del progresso e il motore della vita. Sii sempre curiosa.”
“Curiosity is the soul of progress and the engine of life. Always be curious.”
Prof. Vincenzo Galdi

“True research is born from the humility of knowing you do not know.”
— inspired by Socrates, “I know that I know nothing.”

“The important thing is not to stop questioning.”
Albert Einstein

List of Publications

#	Publication	Type of Publication	In Thesis
1	Paola Vitolo et al. (2023c). "A New NN-Based Approach to In-Sensor PDM-to-PCM Conversion for Ultra TinyML KWS". in: *IEEE Transactions on Circuits and Systems II: Express Briefs* 70.4, pp. 1595–1599. DOI: 10.1109/TCSII.2022.3224022	Journal Article	Chap. 3
2	Paola Vitolo et al. (2024c). "Automatic Audio Feature Extraction for Keyword Spotting". In: *IEEE Signal Processing Letters* 31, pp. 161–165. DOI: 10.1109/LSP.2023.3346280	Journal Article	Chap. 3
3	Paola Vitolo et al. (2022a). "Low-Power Detection and Classification for In-Sensor Predictive Maintenance Based on Vibration Monitoring". In: *IEEE Sensors Journal* 22.7, pp. 6942–6951. DOI: 10.1109/JSEN.2022.3154479	Journal Article	Chap. 4
4	Paola Vitolo et al. (2025). "Real-time neural network-based thermal stress compensation for pressure sensors in precision localization systems". In: *Microprocessors and Microsystems* 117, p. 105183. ISSN: 0141-9331. DOI: https://doi.org/10.1016/j.micpro.2025.105183	Journal Article	Chap. 5
5	Danilo Pau et al. (2023). "Tiny Machine Learning Zoo for Long-Term Compensation of Pressure Sensor Drifts". In: *Electronics* 12, 4819.23. ISSN: 2079-9292. DOI: 10.3390/electronics12234819	Journal Article	Chap. 5

#	Publication	Type of Publication	In Thesis
6	Paola Vitolo et al. (2022b). "Quantized ID-CNN for a Low-power PDM-to-PCM Conversion in TinyML KWS Applications". In: *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 154–157. DOI: 10.1109/AICAS54282.2022.9869909	Conference Proceeding	Chap. 3
7	Paola Vitolo et al. (2023b). "A 0.8 mW TinyML-Based PDM-to-PCM Conversion for In-Sensor KWS Applications". In: *Proceedings of SIE 2022*. Ed. by Giuseppe Cocorullo et al. Cham: Springer Nature Switzerland, pp. 146–151. ISBN: 978-3-031-26066-7. DOI: 10.1007/978-3-031-26066-7_23	Conference Proceeding	Chap. 3
8	Paola Vitolo et al. (2023a). "In-sensor neural network for real-time KWS by image processing". In: *Real-time Processing of Image, Depth and Video Information 2023*. Vol. 12571. International Society for Optics and Photonics. SPIE. DOI: 10.1117/12.2665545	Conference Proceeding	Chap. 3
9	Paola Vitolo et al. (2021). "Low-Power Anomaly Detection and Classification System based on a Partially Binarized Autoencoder for In-Sensor Computing". In: *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pp. 1–5. DOI: 10.1109/ICECS53924.2021.9665479	Conference Proceeding	Chap. 4
10	F. Spinelli et al. (2023). "Low-complexity Machine Learning Architecture for Hardware-aware True Random Number Generators Assessment and Continuous Monitoring". In: *2023 18th Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*, pp. 221–224. DOI: 10.1109/PRIME58259.2023.10161903	Conference Proceeding	Chap. 4

#	Publication	Type of Publication	In Thesis
11	Tommaso Addabbo et al. (2024). "Monitoring Hardware True Random Number Generators with Artificial Neural Networks: Problem Modeling and Training Dataset Generation". In: *Applications in Electronics Pervading Industry, Environment and Society*. Ed. by Francesco Bellotti et al. Cham: Springer Nature Switzerland, pp. 291–296. ISBN: 978-3-031-48121-5	Conference Proceeding	Chap. 4
12	Paola Vitolo et al. (2024a). "In-Sensor System for Real-Time Compensation of Thermal Drift in MEMS Pressure Sensors". In: *Proceedings of SIE 2023*. Ed. by Carmine Ciofi and Ernesto Limiti. Cham: Springer Nature Switzerland, pp. 186–191. ISBN: 978-3-031-48711-8	Conference Proceeding	Chap. 5
13	Paola Vitolo et al. (2024b). "In-Sensor Self-Calibration Circuit of MEMS Pressure Sensors for Accurate Localization". In: *2024 27th Euromicro Conference on Digital System Design (DSD)*, pp. 582–587. DOI: 10.1109/DSD64264.2024.00083	Conference Proceeding	Chap. 5
14	Paola Vitolo et al. (2023d). "Tiny compensation of pressure drift measurements due to long exposures to high temperatures". In: *2023 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pp. 01–05. DOI: 10.1109/I2MTC53148.2023.10175998	Conference Proceeding	Chap. 5
15	Gian Domenico Licciardo et al. (2023). "Ultra-Tiny Neural Network for Compensation of Post-soldering Thermal Drift in MEMS Pressure Sensors". In: *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5. DOI: 10.1109/ISCAS46773.2023.10181480	Conference Proceeding	Chap. 5
16	Danilo Pietro Pau et al. (2024b). "Ultra Tiny Neural Network for Accurate Pressure Sensors Under Multiple Thermal Stresses". In: *2024 IEEE 8th Forum on Research and Technologies for Society and Industry Innovation (RTSI)*, pp. 607–612. DOI: 10.1109/RTSI61910.2024.10761275	Conference Proceeding	Chap. 5

#	Publication	Type of Publication	In Thesis
17	Danilo Pietro Pau et al. (2024a). "Ultra-Tiny Quantized Neural Networks for Piezo Pressure Sensors". In: *2024 Smart Systems Integration Conference and Exhibition (SSI)*, pp. 1–8. DOI: 10.1109/SSI63222.2024.10740508	Conference Proceeding	Chap. 5
18	Paola Vitolo et al. (2024d). "Natural Language to Verilog: Design of a Recurrent Spiking Neural Network using Large Language Models and ChatGPT". in: *2024 International Conference on Neuromorphic Systems (ICONS)*, pp. 110–116. DOI: 10.1109/ICONS62911.2024.00024	Conference Proceeding	Chap. 6

Contents

List of Figures

List of Tables

Abstract

The explosive growth of IoT devices demands on-sensor intelligence that is accurate and radically energy-efficient. This dissertation investigates In-Sensor Computing (ISC) through a constraints-first hardware–software co-design methodology to realize tiny, ultra-low-power neural accelerators tightly coupled to MEMS sensors. The approach shapes network topology, quantization, and fixed-point arithmetic from the outset to meet stringent limits in area, power, and memory, and validates designs through FPGA prototyping and CMOS synthesis. Three application-driven case studies, developed in collaboration with STMicroelectronics, substantiate the methodology. The first addresses audio processing for keyword spotting, where a learned 1D-CNN replaces the conventional CIC+FIR PDM-to-PCM chain, fusing filtering and decimation and delivering 8-bit/16 kHz PCM with 48 dB SNR while preserving downstream accuracy of 89%. The synthesized core in 130 nm CMOS achieves 128.7 µW/MHz within less than 1 mm^2. The second focuses on vibration-based predictive maintenance, employing a hybrid, event-driven pipeline that combines an always-on, partially binarized in-sensor autoencoder for anomaly detection (AUC = 0.99; 99.61% accuracy) with an on-demand MCU classifier (up to 94.83%). The in-sensor accelerator sustains sensor output data rates up to 365 kHz and exhibits 333 µW/MHz dynamic power on FPGA, while standard-cell synthesis in 65 nm reports 0.49 mm^2 and 138.6 µW/MHz dynamic power. The third case concerns thermal-stress compensation for MEMS pressure sensors: the proposed AI-based Reconfigurable Sensor Compensation Unit (AI-ReSCU) couples a reconfigurable trigger with an iterative neural error estimator with binarized weights and fixed-point activations to restore accuracy within ±0.5 hPa, recovering up to 1.6 hPa, with 4.46 nW dynamic power in 0.55 mm^2. Taken together, these diverse studies confirm the general applicability of the proposed design flow: despite their different sensing domains and performance targets, each achieves state-of-the-art accuracy and efficiency. The dissertation distills generalizable ISC design principles—early constraint propagation, aggressive resource sharing with serialized compute, selective binarization and low-bit quantization, and event-triggered operation with deep sleep—showing that competitive machine-learning accuracy and real-time throughput can be achieved at milliwatt-to-nanowatt power and sub-mm^2 area, enabling practical in-sensor AI. Finally, during a 6-month research period at Johns Hopkins University, an exploratory study investigated Large Language Models (LLMs)-assisted hardware-description generation, including synthesizable Verilog, testbenches, and documentation for a recurrent spiking neural network validated on FPGA and implemented with an open-source SkyWater 130 nm flow, as a complementary perspective on how the proposed co-design workflow could be accelerated.

Introduction

The Internet of Things (IoT) is revolutionizing the way the digital and physical worlds interact, connecting billions of devices—ranging from industrial machines to consumer electronics—into vast, data-rich networks. These devices continuously sense, collect, and exchange data, enabling advanced automation, real-time monitoring, and intelligent decision-making across a wide range of domains. However, the explosive growth of IoT, both in terms of device count and data volume, is introducing significant challenges in data processing, storage, communication, and energy consumption.

According to recent estimates, over 27 billion IoT devices are expected to generate more than 73 zettabytes (ZB) of data annually by 2025. This massive volume of information is impractical to transmit and process entirely in centralized cloud infrastructures. The energy required for wireless communication, the bandwidth limitations of networks, and increasing concerns over data privacy all call for new paradigms in distributed computing.

A promising solution to these challenges is *Edge Computing (EC)*, where data is processed closer to its source. EC reduces latency, lowers energy consumption, and improves privacy by enabling real-time decision-making at the device level. Going a step further, *In-Sensor Computing (ISC)* represents the most granular form of edge computing: it integrates computational capabilities directly into the sensor package. In this paradigm, sensors not only detect and transduce physical signals but also perform part of the processing required to extract meaningful information before it ever leaves the sensor.

ISC is made possible by decades of technological scaling. Advances in CMOS technology have drastically reduced the size and power consumption of transistors, while MEMS (Micro-Electro-Mechanical Systems) sensor technology has evolved to fit complex sensing functions within millimeter-scale packages. This progress enables the integration of custom, low-power digital circuits near or within sensors, paving the way for real-time, localized intelligence.

Despite its potential, ISC introduces new and stringent constraints. Power budgets are often limited to a few hundred microwatts—or even nanowatts in energy-critical applications—while silicon area is tightly constrained, typically less than a few square millimeters. These limitations stand in stark contrast to the computational demands of modern *Neural Networks (NNs)*, which have proven highly effective for a wide range of data-driven tasks, but were originally developed for power-hungry platforms such as GPUs and data centers.

This research aims to bridge the gap between the power of neural networks and

the severe constraints of in-sensor computing. The objective is to design *tiny, ultra-low-power hardware accelerators* for neural networks that can operate within the tight area, energy budget, and computational capabilities of ISC environments.

To achieve this, the project adopts a *hardware–software co-design* methodology, in which the neural model and hardware are developed in tandem. This approach ensures that architectural decisions—such as network topology, quantization levels, and computational precision—are made with full awareness of hardware limitations from the very beginning. Such co-design is essential to produce solutions that are not only functionally accurate but also physically realizable at the edge.

The design flow consists of five main phases:

- **Requirements and specifications definition**, where application-level targets and sensor-integration constraints (power, area, memory, latency, interfaces) are formalized and propagated through the subsequent stages;
- **Data acquisition and pre-processing**, where raw sensor data is collected and cleaned to create meaningful input features;
- **Neural network modeling**, where different NN architectures are designed and optimized using quantization-aware training and compression techniques;
- **Hardware design**, where the optimized model is translated into a digital accelerator using hardware description languages;
- **Implementation and validation**, where the design is prototyped on FPGAs and synthesized on CMOS standard-cell technology to verify performance and feasibility near sensor terminals.

Crucially, these steps are not independent. The methodology follows a co-design loop, where hardware feedback is continuously used to refine the model. If the hardware implementation does not meet the power or area constraints, the model is revisited and re-optimized. This tight coupling between software and hardware enables efficient exploration of the design space under real-world constraints.

The proposed methodology was applied to three case studies, developed in collaboration with *STMicroelectronics*:

1. **Keyword Spotting (KWS)** using digital MEMS microphones. In this case, a compact neural network replaces traditional Pulse Density Modulation (PDM) to Pulse Code Modulation (PCM) converters. The system achieves competitive accuracy and low latency, with a power consumption of only 128 μW/MHz and an area of 0.09 mm^2 in 130 nm CMOS technology.
2. **Anomaly Detection (AD)** for Predictive Maintenance using accelerometers. A hybrid architecture combines a binarized autoencoder for anomaly detection with a lightweight Convolutional Neural Network (CNN) for classification. The CNN is only activated when anomalies are detected, minimizing average power consumption while maintaining high diagnostic accuracy.
3. **Self-calibration of MEMS pressure sensors**. Here, a neural network is used to dynamically compensate for thermal drift affecting pressure readings. The calibration unit is implemented as a dedicated ASIC, achieving real-time correction with a power consumption of just 4.46 nW and a silicon area of 0.55 mm^2.

These three case studies were selected because they originate from concrete industrial requirements and, together, offer a representative cross-section of ISC scenarios where an *always-on* sensing node must extract actionable information under tight integration constraints. They span different sensor modalities (microphone, accelerometer, and pressure sensor), different signal time constants (from high-rate streaming to slow thermal dynamics), and different objectives (audio front-end processing, anomaly monitoring, and self-calibration). This diversity provides a meaningful testbed to apply the same constraints-first hardware–software co-design methodology across heterogeneous contexts and to assess its generality and transferability.

Finally, this thesis includes an *exploratory* study on the use of Large Language Models (LLMs) as a productivity aid for hardware development. During a six-month research period at Johns Hopkins University, an LLM was used to assist the generation of synthesizable Verilog, testbenches, and documentation for a recurrent spiking neural network design, validated on FPGA and implemented with an open-source SkyWater 130 nm flow. This study provides a complementary perspective on how selected steps of the hardware-development workflow could be accelerated, highlighting the growing potential of AI not only as an application domain but also as a design tool.

This thesis is organized as follows:

- **Chapter 1** provides a state-of-the-art overview of in-sensor computing and neural networks for edge applications, identifying gaps and challenges in the current literature;
- **Chapter 2** describes the adopted methodology, including model development, optimization techniques, hardware implementation, and the co-design approach;
- **Chapters 3 through 5** present the three case studies, each detailing the problem context, model design, hardware implementation, and experimental results;
- **Chapter 6** discusses the main findings, cross-case insights, and limitations of the proposed approach;
- **Conclusions** summarizes the results and outlines future directions, including opportunities for AI-assisted design workflows and next-generation smart sensors.
- **Appendix A** reports an exploratory study on AI-assisted hardware design using Large Language Models (LLMs), to assist in the design, implementation, and verification of hardware accelerators. It describes the development of a spiking neural network accelerator using automatically generated Verilog code, testbenches, and documentation, and evaluates the effectiveness of this AI-assisted approach;

Through these contributions, the research demonstrates a concrete pathway toward the development of smarter and energy-efficient sensors, addressing the computational demands of tomorrow's IoT world at the sensor level.

The research presented in this thesis was carried out between 2021 and 2025 and has been disseminated through peer-reviewed journals and international conferences. Specifically, it has resulted in 13 conference papers, 3 articles in Q1 journals, and 2 in Q2 journals. References to these works are provided throughout the thesis, and the complete list of publications is reported at the beginning of this manuscript.

Chapter 1
Survey of Architectures and Techniques for Edge and In-Sensor Neural Computing

This chapter provides a comprehensive review of the state of the art relevant to in-sensor computing and neural networks for extreme edge applications.

While the Introduction of this thesis has already outlined the overarching motivations for exploring ISC—energy efficiency, minimal latency, and on-device privacy—the discussion here is on the principal technological approaches and their current limitations. As highlighted in the Introduction, integrating processing directly within the sensing element enables the creation of smart sensors with greater functionality and enhanced performance.

By extracting only the most relevant information at the source, such integration reduces the volume of raw data to be transmitted, thereby lowering transmission power, bandwidth usage, and latency. It also reinforces on-device privacy and security, reduces data-storage requirements, and can improve the accuracy and overall functionality of the sensors themselves, enabling more effective downstream processing.

Building on these motivations, the present chapter focuses on the technological approaches proposed to realize ISC and on the limitations that still hinder their adoption. The first part reviews architectural concepts and enabling technologies that allow computation to be embedded inside a sensor package, highlighting representative applications and the severe constraints on area and power that shape hardware design.

The second part examines the landscape of neural-network solutions for edge devices, with attention to lightweight model families, compression and quantization strategies, and hardware platforms capable of meeting extreme resource constraints.

This analysis provides the broader context for the hardware–software co-design methodology adopted in this research work, showing why dedicated neural hardware accelerators are essential for practical ISC. Detailed, domain-specific state-of-the-art reviews are intentionally deferred to the individual case-study chapters, where each application is examined within its own specialized literature.

I.1 Principles and Enabling Technologies for In-Sensor Processing

In-sensor processing (also called *processing-in-sensor*) refers to the integration of computational capabilities directly within a sensor device or package, so that data can be partially or fully processed at the point of capture rather than transmitted raw. The goal is to dramatically reduce the data volume and latency between sensing and inference, improving energy efficiency and enabling more intelligent ”smart sensors” at the extreme edge. As discussed in the Introduction, moving computation closer to the sensor can alleviate the bandwidth and energy bottlenecks of shuttling large volumes of raw data to a separate processor. It also improves on-device privacy by avoiding transmission of sensitive raw signals. In practice, however, realising effective in-sensor computing requires careful co-design of algorithms and hardware under severe area and power constraints, as the logic must fit alongside or within the sensing elements. This section reviews the principal architectural paradigms for in-sensor processing and the enabling circuit technologies.

I.1.1 Near- and In-Sensor Computing Architectures

Researchers have explored several architectural approaches to push neural network computations closer to the sensor, which can be broadly categorized as *near-sensor*, *in-sensor*, or *in-pixel* processing, as illustrated in Fig. I.1 (Datta et al., 2022).

Near-sensor processing places a dedicated processor or accelerator adjacent to the sensor—on the same board or in a 3D-stacked package—so that raw data need not travel to a remote edge device (Pinkham et al., 2021). This shortens interconnects and cuts transfer overheads. For example, Pinkham et al. (2021) report a 64% energy reduction when a low-power CNN accelerator is co-packaged with an image sensor for MobileNet inference. A notable industrial case is Sony’s *Intelligent Vision Sensor* IMX500, which bonds a pixel array to a logic layer hosting a digital signal processor, achieving up to ∼5 TOPS/W for CNN inference (Eki et al., 2021). Such 3D stacking illustrates how computation can be performed “next to” the sensor, avoiding the high latency and power cost of streaming full-resolution data.

In **in-sensor processing**, the sensor ASIC itself integrates analog or digital circuits

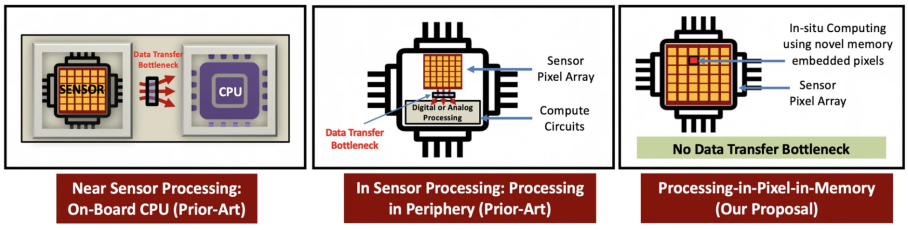

Figure I.1: *Comparison of sensing–compute architectures from (Datta et al., 2022): (left) near-sensor processing with an on-board CPU, (center) in-sensor processing using periphery circuits, and (right) the proposed processing-in-pixel-in-memory approach that eliminates data-transfer bottlenecks. Image extracted from (Datta et al., 2022).*

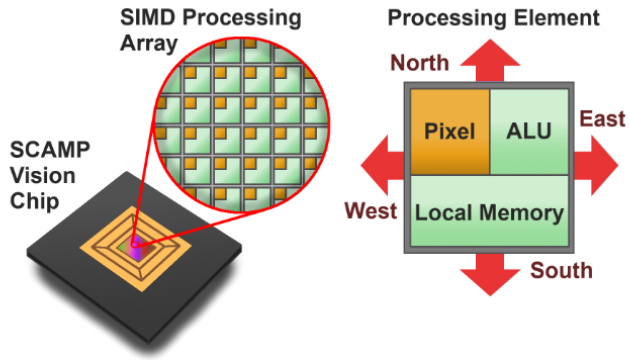

Figure I.2: *The Pixel Processor Array described in (L. Bose et al., 2020) executes computations directly on the image sensor chip through a SIMD processor grid, where each pixel integrates an arithmetic logic unit (ALU), local memory, and connections for communication with adjacent pixels. Image extracted from (L. Bose et al., 2020).*

to process signals before they leave the chip (Z. Chen et al., 2020; LiKamWa et al., 2016). This approach can output only high-level features or classifications, dramatically reducing data transfer and ADC power. Early work such as RedEye performed the first convolutional layers of GoogLeNet directly in the analog front-end, cutting per-frame energy by over $5\times$ (LiKamWa et al., 2016). Other designs embed small CNNs in the sensor's readout path to perform tasks such as face detection at sub-volt operation (Hsu et al., 2022). Most implementations rely on column-parallel circuits at the array periphery rather than true per-pixel compute, which limits bandwidth for very high resolutions but still removes the need to digitize and transmit redundant raw data.

In-pixel processing pushes computation to the pixel level, so that each pixel contains not only the photodiode but also minimal logic or analog devices to process its own signal, as shown in Fig. I.2 (L. Bose et al., 2020). Digital pixel-processor arrays (PPAs) execute SIMD operations across a grid of tiny ALUs, enabling real-time filtering or shallow CNN layers, albeit with reduced resolution or low-bit precision due to area constraints. Analog approaches exploit device physics for multiply–accumulate operations using elements such as 2D semiconductor phototransistors (Mennel et al., 2020) or memristive cells (Ouyang et al., 2024), achieving prototype energy efficiencies up to 10–17 TOPS/W (Song et al., 2022). These concepts promise extreme parallelism and minimal data movement but face challenges of noise, limited precision, and integration with standard CMOS.

The use of emerging materials (2D semiconductors, memristors, etc.) means these sensors are often experimental and not compatible with standard CMOS fabrication yet. Moreover, analog computations suffer from noise, device variation, limited precision, and lack of programmability. Many analog in-pixel demonstrations to date handle only binary or few-bit signals or require off-chip training/calibration due to device non-idealities. For instance, some designs encode neural network weights as variable exposure times or pulse widths applied to pixels (Datta et al., 2022), which indeed implements a form of analog multiply but necessitates supplying those con-

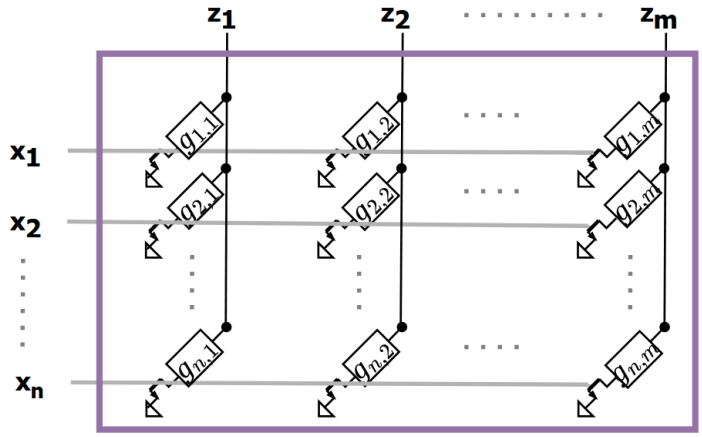

Figure I.3: *Physical crossbar array used for analog in-memory computing (AIMC). Each intersection between rows $x_1 \dots x_n$ and columns $z_1 \dots z_m$ contains a programmable conductance $g_{i,j}$, enabling matrix–vector multiplication by applying input voltages to the rows and collecting output currents from the columns. Image extracted from (Antolini et al., 2025).*

trol pulses from an external memory, partly negating the data reduction benefits. In short, while in-pixel computing (especially analog) offers a tantalizing vision of ultra-efficient co-located sensing and processing, it remains difficult to implement multi-layer, high-accuracy deep networks entirely inside a standard sensor pixel array with current technology.

I.1.2 Technology Drivers and Constraints

The move toward ISC is driven by technology trends that bring computation closer to the sensing plane, even as stringent always-on constraints continue to shape design choices. Advanced CMOS processes, 3D integration, on-sensor memory, increasingly complex analog/digital circuits, and moderate application requirements now make it feasible to shift part of the processing directly inside the sensor. Below we highlight the key drivers and the core constraints that shape practical ISC solutions.

Technology drivers enabling ISC:

- **Advanced CMOS process scaling and 3D stacking** – Smaller CMOS nodes and stacked sensor–logic architectures (e.g., backside-illuminated pixel arrays bonded to a separate logic die) create extra silicon area for neural accelerators and control circuits without compromising optical performance.

- **On-sensor memory integration** – Embedded SRAM or emerging non-volatile memories can be placed beneath or alongside the pixel array, enabling local storage of neural-network weights and temporary buffers for feature data.

- **Mature active-pixel sensors and low-power ADCs** – Modern APS designs support more complex per-column or grouped-pixel circuitry, allowing basic

computation during readout while keeping pixel noise and fill-factor within acceptable limits.

- **More capable analog/digital circuit blocks** – Improvements in low-power analog front-ends and compact digital logic make it practical to embed multiply–accumulate or small neural units directly on the sensor die.
- **Moderate requirements of many edge applications** – Tasks such as vibration monitoring, always-on audio, or event-driven vision often operate at low resolution or frame rate, relaxing area and bandwidth demands and favoring early feature extraction instead of full raw-data transmission.

Key Constraints Governing Practical ISC Implementations:

- **Ultra-low power budgets** – Sensors are often always-on and must operate in the microwatt–milliwatt range. Any integrated compute therefore needs aggressive power management: near- or sub-threshold operation, deep clock gating, event-driven wake-up strategies, and careful minimization of memory access to avoid excessive dynamic power.
- **Limited silicon area and pixel fill-factor** – Logic circuits compete with the light-sensitive photodiode area. Adding transistors can reduce the effective pixel size or degrade sensitivity, so designers often move compute to the column periphery or adopt 3D stacking to reclaim area while preserving optical performance.
- **Precision and robustness trade-offs** – Analog processing can deliver excellent energy efficiency but suffers from noise, device variability, and limited programmability. Digital implementations are more robust and reconfigurable but consume more energy and require larger transistor counts. Many designs adopt a mixed-signal approach to balance these factors.
- **Memory and bandwidth limits** – In-sensor architectures cannot rely on large off-chip DRAM. On-chip buffers must be extremely small, so data reuse, early feature extraction, and streaming computation are critical to keep internal bandwidth and storage demands within feasible limits.
- **Thermal constraints** – Because sensors may be embedded in compact or sealed systems, even modest power dissipation can raise local temperature, affecting both sensor noise and device reliability. Thermal considerations strongly influence clock frequency, supply voltage, and duty cycling.

I.1.3 Analog vs. Digital In-Sensor Computing

In practice, *analog* vs. *digital* in-sensor computing boils down to complementary strengths and hard constraints:

- **analog** excels at ultra-low-energy MACs and natural parallelism by operating before the ADC, but pays in precision, variability, and limited programmability (LiKamWa et al., 2016; Ma et al., 2019);

- **digital** offers robustness, exactness for a given bit-width, and a mature tool/IP ecosystem, but must tame switching and memory power via quantization and model compression (Ray, 2022);

- **hybrid** partitioning leverages analog where energy-per-operation is unbeatable and uses digital for non-linearities, memory, and control, as in processing-in-pixel(-in-memory) schemes (Datta et al., 2022).

Analog approaches exploit the fact that signals are born analog: charge sharing and current summation can implement convolutions with orders-of-magnitude lower energy per operation, while massively parallel operation across the array minimizes data movement (LiKamWa et al., 2016). The price is sensitivity to noise and PVT variations, tight precision (often 1–4 bit for weights/activations), and reduced flexibility—especially when specialized devices (e.g., memristors or 2D materials) are involved and not yet mainstream in CMOS image-sensor flows (Mennel et al., 2020; Ouyang et al., 2024). Digital compute, by contrast, cleanly interfaces with existing readout chains (or even consumes a $\Sigma\Delta$ bitstream directly in certain sensors), scales with standard CMOS, and remains reprogrammable and verifiable; yet, clocks, buses, and memories dominate energy unless networks are co-designed for tiny footprints (low-bit quantization, depthwise separable kernels, pruning) (Ray, 2022). A pragmatic consensus is thus a mixed-signal co-design: perform early, high-parallelism MACs or feature extraction in the analog domain to curb ADC and bandwidth, then hand off to lean digital blocks for ReLU/BN, control, and final classification—an approach validated by recent processing-in-pixel(-in-memory) demonstrations (Datta et al., 2022) and by commercial "smart sensors" that pair analog front-ends with small digital cores for on-package inference.

In summary, the state of the art in edge AI sensors spans a spectrum from purely analog in-sensor computing to fully digital on-sensor processors, with various intermediate hybrids. Each approach must contend with the stringent area and power limits inherent to sensor-integrated circuits. The review above shows that embedding neural network processing at the sensor is not only conceptually possible, but already yielding working prototypes and even initial products. However, it also underscores why a careful hardware-software co-design is essential: one must tailor the neural network (architecture, quantization, etc.) to the constraints of the sensor hardware, and conversely design the hardware to efficiently support the operations that the neural algorithm actually needs. This co-design philosophy is a central theme of this thesis.

Given the practical considerations, our work gravitates toward the digital end of the spectrum, using traditional CMOS design flows to create extremely compact neural network accelerators that can be integrated with sensors. The rationale is that digital circuits in scaled CMOS provide the most predictable, reproducible behavior and are readily synthesizable, which is advantageous for industrial adoption. While analog and exotic-device approaches are promising, their uncertainties in fabrication and algorithmic generality are higher at this stage. By focusing on digital in-sensor computing, we leverage existing technology (standard CMOS, standard cell libraries, SRAM, etc.) to implement neural network inference engines that operate within micro-Watt power budgets. Chapters to follow will detail how we design such accelerators and corresponding tiny neural networks for various sensing applications (from vibration

monitoring to pressure sensor compensation). Each case study will demonstrate the hardware/software co-design needed to meet the extreme edge constraints, building on the principles and state-of-art techniques surveyed here. Our approach, in line with recent trends, validates that with careful design, a *digital CMOS smart sensor* can significantly enhance functionality (running AI algorithms on raw sensor data) while respecting the tight power and area envelopes of embedded sensing platforms. This confirms that digital in-sensor computing is not only an attractive concept but a feasible solution for next-generation ultra-efficient edge intelligence.

I.2 Neural Network Design Techniques and Hardware Platforms for Deployment on Edge Devices

Deploying neural networks on resource-constrained edge devices (such as microcontrollers and IoT sensors) requires model and hardware innovations beyond those used in power-rich cloud environments. Edge devices operate under strict limitations in memory, compute capacity, and energy supply. Accordingly, the state of the art in *TinyML* (tiny machine learning) has evolved a toolkit of strategies to enable efficient on-device inference while maintaining acceptable accuracy. This section reviews these strategies at both the model level and the hardware level, and outlines the standard TinyML development flow that has inspired the custom ASIC-oriented methodology proposed in this thesis.

I.2.1 Model-Level Strategies for Edge Deployment

One approach to meeting edge constraints is **lightweight neural network architecture design**. Over the past few years, researchers have proposed neural architectures explicitly optimized for low compute and memory footprint. A seminal example is SqueezeNet, a small CNN that achieves AlexNet-level accuracy with $50\times$ fewer parameters (model size $<$0.5 MB) by using bottleneck "fire" modules with 1×1 convolutions (Iandola et al., 2016). Another prominent family is MobileNet, which introduced depthwise separable convolutions to drastically reduce the number of multiplications per inference (Howard et al., 2017). MobileNet-v2 further improved on this with inverted residual blocks and linear bottlenecks, allowing even deeper networks at a fraction of the cost of conventional CNNs (Sandler et al., 2018). These design patterns (depthwise separable filters, bottlenecks, etc.) have become standard in TinyML-friendly models. For extreme resource budgets, researchers have also explored binarized or ternarized neural networks, where weights (and sometimes activations) are constrained to ±1 or $-1, 0, +1$, vastly simplifying the required arithmetic operations (Yannan Wang et al., 2018; S. Woo et al., 2024). Such aggressively quantized models can replace expensive 32-bit multiply-accumulate operations with bit-wise operations, though often at some loss in accuracy.

In addition to manually designed small models, **neural architecture search (NAS)** has emerged as a powerful tool for discovering efficient networks tailored to edge devices. NAS methods automate the exploration of architecture configurations under resource constraints (e.g. limiting total operations, memory footprint or latency). For instance, MnasNet was obtained via a platform-aware NAS that incorporated real smartphone latency into the reward function, yielding a model that is $1.8\times$ faster than MobileNet-v2 for similar accuracy (Tan et al., 2019). Facebook's FBNet similarly used a differentiable NAS approach to find networks optimized for hardware efficiency (B. Wu et al., 2019). More recent efforts like *TinyNAS* (part of the MCUNet framework) explicitly target microcontroller limitations, co-designing the search space and inference engine to fit within a few hundred kilobytes of memory (Lin et al., 2020a). There are also curated collections of efficient "micro architectures" produced by community challenges; for example, the MicroNets project evaluated numerous tiny models on common microcontroller tasks, yielding a set of architectures that push the Pareto front of accuracy vs. resource usage for TinyML applications (Banbury et al.,

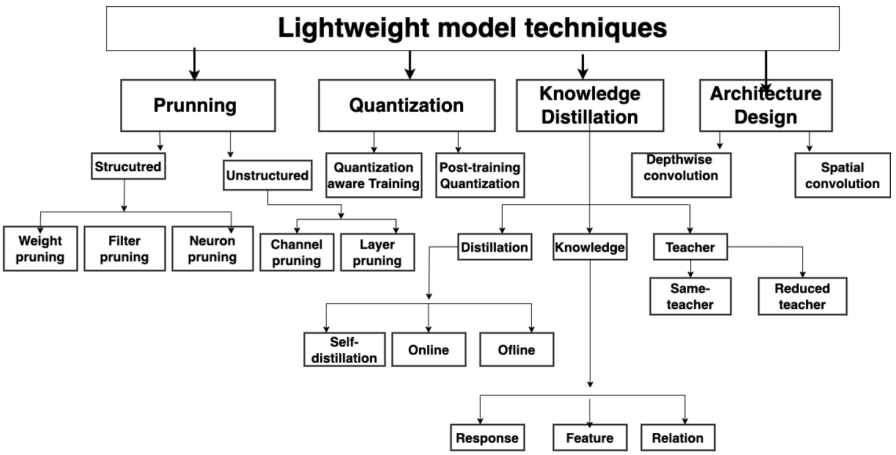

Figure I.4: *Overview of lightweight deep learning model techniques, illustrating key strategies such as pruning, quantization, knowledge distillation, and architecture design, along with their principal subcategories and hierarchical relationships. Image extracted from (Musa et al., 2025)*

2021). Across these approaches, the guiding principle is to maximize model compactness and efficiency — in terms of parameter count, operation count, and memory access — while preserving as much task performance as possible. Techniques like layer width scaling, reduced kernel sizes, and operation fusion are commonly employed. Early profiling on target hardware or simulators is often used during design to ensure the network meets the intended latency and memory targets (Lin et al., 2020a).

Beyond architectural choices, there is a rich toolkit of **model compression and optimization techniques**, summarized in Fig. I.4 (Musa et al., 2025), to further reduce the footprint of neural networks for edge deployment.

Quantization is one of the most effective: by reducing the precision of network parameters and activations from 32-bit floating point to 8-bit (or even lower) integers, the model's memory usage and compute cost can be cut dramatically. Integer arithmetic is not only faster on many embedded processors, but also yields lower power consumption. Post-training quantization (PTQ) can be applied to a pre-trained model, converting weights to 8-bit integers and optionally calibrating activations on a calibration dataset. This typically incurs only a small loss in accuracy for many vision and audio models, while yielding a $4\times$ reduction in model size and similarly large speedups by leveraging efficient integer arithmetic (Jacob et al., 2018). For example, Jacob et al. (2018) showed that an 8-bit quantized MobileNet can run significantly faster on mobile CPUs with only minor accuracy degradation. For even better accuracy retention, *quantization-aware training* (QAT) can be used: here, the model is trained (or fine-tuned) with fake quantization nodes in the graph so that the network learns to operate under low-precision constraints. QAT tends to outperform PTQ in accuracy, especially when quantizing below 8-bit, at the cost of a more complex training procedure (Jacob et al., 2018). Tools like QKeras provide convenient layers for QAT, enabling training with mixed precision or ultra-low precision (e.g. 4-bit, 2-bit) while accounting for hardware deployment constraints (Coelho et al., 2021).

Another major approach is pruning, which eliminates redundant parameters or structures from the model. By removing less impactful weights (unstructured pruning) or even entire neurons/filters (structured pruning), one can obtain a sparser network that computes faster and uses less memory. Han et al. (2015b) demonstrated that iterative pruning combined with fine-tuning can dramatically compress models – for example, pruning yielded $9\times$ fewer parameters on AlexNet with no loss in accuracy, and $13\times$ fewer on VGG-16 with only 0.4% accuracy drop (Han et al., 2015b). Pruning works synergistically with quantization: after pruning, the remaining weights can be quantized for additional gains, as shown in the "Deep Compression" framework Han et al., 2015b. Modern pruning techniques often aim for structured sparsity (removing whole filters or channels) so that the resulting model can still exploit efficient dense operations on hardware (Molchanov et al., 2017). There is ongoing research into hardware accelerators that can take advantage of fine-grained unstructured sparsity as well, although irregular memory access patterns can limit the speedups on general-purpose processors.

A complementary technique to reduce model complexity is knowledge distillation. In distillation, a large, high-accuracy "teacher" model (or an ensemble of models) is used to guide the training of a much smaller "student" model (Hinton et al., 2015). Rather than training the student solely on the original dataset labels, the student is also trained to match the teacher's soft output probabilities (or intermediate representations). These softer targets provide richer information, helping the small model mimic the teacher's function. Knowledge distillation has proven effective in boosting the accuracy of compact models without increasing their size or latency. For edge scenarios, one might train a big network offline (e.g. a ResNet on a GPU cluster) and then distill its knowledge into a tiny network suitable for a microcontroller. The result is often a significant improvement in the student model's accuracy compared to training it from scratch or with naive label training, closing some of the performance gap due to model size reduction (Hinton et al., 2015). Distillation can be combined with the other techniques above: for instance, one could quantize a model and then use a teacher to fine-tune it via distillation, or distill a large model into a smaller architecture that was found by NAS. By mixing and matching architecture innovation, quantization, pruning, and distillation, state-of-the-art TinyML pipelines are now capable of running complex tasks (image classification, keyword spotting, anomaly detection, etc.) on tiny Cortex-M class microcontrollers within tight memory (tens of kilobytes of RAM/Flash) and latency (tens of milliseconds) budgets (Banbury et al., 2021; Lin et al., 2020a). Notably, these optimizations typically involve iterative refinement: developers may alternate between training (or fine-tuning) the model with constraints and evaluating on hardware simulators or profilers to ensure the design will meet the deployment requirements.

I.2.2 Hardware-Level Strategies and Edge AI Accelerators

Model improvements alone are often not sufficient – efficient **hardware implementation** is equally crucial for edge neural networks. Contemporary approaches span from optimizing software on existing microcontroller units (MCUs) to designing specialized neural accelerators. At the simplest level, software libraries can greatly improve the efficiency of neural network inference on commodity microcontrollers. A

prime example is ARM's CMSIS-NN library, which provides highly optimized fixed-point kernels for common operations like convolutions, fully-connected layers, and activations (Lai et al., 2018). By carefully implementing these functions in assembly to utilize the processor's DSP instructions and parallelism, CMSIS-NN achieves substantial speedups (e.g. 4.6× faster inference and nearly 5× lower energy for a suite of benchmarks, compared to baseline C implementations) (Lai et al., 2018). Such low-level optimizations take advantage of hardware features (like 8-bit SIMD operations and specialized MAC instructions) that are often underutilized by general-purpose compilers. In addition to CMSIS-NN (which targets Arm Cortex-M cores), many microcontroller vendors provide their own inference libraries or SDKs (e.g. NVIDIA's TensorRT for their Jetson SoCs, or Cadence's optimized libraries for Tensilica DSPs), aiming to squeeze out every drop of performance. On top of these kernels, lightweight inference runtimes such as *TensorFlow Lite Micro* abstract the deployment process: TFLite Micro takes a trained model and handles tasks like tensor allocation, quantized operator invocation, and platform abstraction, all within a few kilobytes of code overhead (David et al., 2021). It is designed to run without dynamic memory allocation or operating system support, matching the constraints of bare-metal MCU environments. Using such frameworks, developers can deploy a neural network (after quantization) onto devices with as little as 32KB of RAM. Other frameworks like *µTVM* (micro TVM) (Y.-H. Chen et al., 2017) and vendor tools (e.g. STMicroelectronics' STM32Cube.AI code generator) serve a similar purpose: they take a high-level model and generate highly optimized C/C++ or assembly code for a specific target device, often integrating with the vendor's IDE and toolchain. These tools also perform ahead-of-time analysis of memory usage (to statically plan tensor buffers in limited SRAM) and check that the model's layers are supported on the target device. By deploying on actual hardware or cycle-accurate simulators, developers can profile the inference to measure latency (e.g. in ms) and energy consumption (e.g. via on-board power monitors), ensuring the model meets the real-time requirements of the application (Reddi et al., 2021).

For use cases that demand orders-of-magnitude higher performance or efficiency than general MCUs can offer, designers turn to **specialized edge AI accelerators**. These range from neural processing units (NPUs) integrated into system-on-chip devices (for example, the ARM Ethos-U microNPU or the NPU in modern smartphone SoCs) to standalone accelerator chips like Google's Edge TPU. The fundamental idea is to provide dedicated hardware datapaths for the common operations in neural networks (matrix multiplies, convolutions, etc.), optimized for throughput and energy efficiency. By removing the overhead of general-purpose instruction processing and tailoring memory hierarchies to neural network dataflows, accelerators can achieve far better energy-per-inference. For instance, the Eyeriss accelerator demonstrated efficient convolutional neural network processing with a spatial array of processing elements, achieving significant energy savings by exploiting data reuse and minimizing off-chip memory access (Y.-H. Chen et al., 2017). In general, off-chip DRAM access is extremely energy-expensive relative to on-chip computation, so many accelerators employ scratchpad SRAM buffers and smart scheduling to reuse weights and activations as much as possible (Y.-H. Chen et al., 2017; Sze et al., 2017). Architectures like Eyeriss and others in the literature make heavy use of parallelism (hundreds of small multiply-accumulate units operating simultaneously) and customized interconnects to

keep data flowing efficiently between memory and compute units. The result can be measured in tera-operations per second per Watt (TOPS/W) – a common metric for efficiency. Modern edge AI chips achieve on the order of 1–10 TOPS/W, which is an order of magnitude higher than typical microcontroller efficiency. Google's Edge TPU, for example, delivers up to 4 trillion operations per second in a small module powered by only a few watts, by leveraging an array of 8-bit multiply-add units and an on-chip memory optimized for conv layer access patterns.

While these accelerators provide raw speed, they introduce **design trade-offs in energy, latency, and area**. A larger accelerator array can compute more operations in parallel, reducing latency for a given inference, but it consumes more silicon area and typically more power (especially dynamic power from switching many units each cycle). Conversely, a smaller or more power-gated design might save energy at the expense of higher latency or lower peak throughput. In edge scenarios, the optimal design often depends on the application requirements: "always-on" inference tasks (like wake-word detection or continuously monitoring a sensor) prioritize ultra-low energy per inference so that the device can run on battery or harvested energy indefinitely, whereas less frequent but compute-intensive tasks might tolerate a bit more energy per inference for faster results. Designing accelerators thus becomes a multi-objective optimization: minimize energy and area while meeting a certain latency target. Techniques such as voltage/frequency scaling, power gating, and clock domain separation are used to strike this balance in hardware (Sze et al., 2017). Moreover, since memory accesses dominate energy cost, many accelerators use model sparsity and compression to reduce the data volume moved on- and off-chip (e.g., skipping zero weights or using weight compression formats). Some digital accelerators even support *dynamic voltage and frequency scaling* (DVFS) or operate in near-threshold voltage regimes to further cut power, though this can introduce variability in performance. Beyond digital designs, there are emerging analog and in-memory computing approaches to accelerate neural networks; however, these are still largely experimental and face challenges with precision and flexibility. For mainstream edge AI in 2025, digital accelerators optimized for 8-bit integer operations remain the prevalent solution. The diversity of hardware – from 32-bit microcontrollers with DSP extensions, to NPUs in IoT SoCs, to tiny always-on ASICs – means that TinyML model design must often be hardware-aware, tailoring the neural network to exploit the strengths and mitigate the weaknesses of the deployment platform.

I.2.2.1 Commercial NPUs and Vendor Accelerators

Beyond MCU-only deployments, commercial embedded platforms increasingly integrate dedicated neural accelerators (NPUs) to improve throughput and energy efficiency for low-precision inference. A representative example is the *ST Neural-ART Accelerator*, integrated in the STM32N6 family as an on-chip NPU targeted at power-efficient edge AI workloads (STMicroelectronics, 2024b; STMicroelectronics, 2024a). According to ST documentation, Neural-ART is a dataflow-oriented engine supporting 8–16 bit arithmetic, configurable compute resources, and is marketed to deliver up to $\sim$600 GOPS within the STM32N6 device class (STMicroelectronics, 2024b; STMicroelectronics, 2024a). This type of solution exemplifies a broader trend also seen in *microNPUs* coupled to Cortex-M class processors (e.g., Arm Ethos-U

family) (Arm, 2021).

While such accelerators are often highly effective for *near-sensor* or *edge* deployment, their suitability for *true* in-sensor computing (ISC) must be critically assessed. ISC targets impose significantly tighter constraints than typical MCU/SoC scenarios, because the compute must fit within (or be tightly co-packaged with) the sensing element and operate under stringent always-on power envelopes. In particular, four aspects are decisive:

- **Integration footprint and memory hierarchy:** commercial NPUs are typically embedded in systems that assume a relatively rich on-chip SRAM and SoC interconnect. This favors near-sensor processing, whereas ISC often demands sub-mm^2 logic footprints and extremely limited on-sensor memory.
- **Data movement vs. compute energy:** even when the NPU offers a favorable TOPS/W figure, the *system-level* energy can be dominated by moving sensor data into buffers and shuttling activations/weights through SRAM and buses. ISC instead aims to minimize conversions, buffering, and transfers by exploiting early feature extraction and streaming datapaths.
- **Always-on operation and energy proportionality:** in many ISC applications the workload is continuous or event-driven with strict duty-cycling requirements. General-purpose NPUs may achieve high peak efficiency, yet they are not necessarily optimized for ultra-low average power under sparse activation patterns and aggressive clock/power gating.
- **Operator/model constraints vs. specialization:** vendor NPUs and their toolchains typically support a subset of operators and quantization schemes optimized for portability. Conversely, ISC architectures can exploit application-specific fixed models, tensor shapes, and ultra-low-bit quantization to reduce memory and control overhead beyond the constraints of general-purpose deployment flows.

Overall, commercial NPUs provide an essential baseline for practical edge intelligence and can be excellent candidates for *near-sensor* smart systems (e.g., sensors coupled to a local MCU/SoC). However, when the target is *sensor-integrated* inference under extreme area and always-on power budgets, a constraints-first co-design approach and custom architectures become necessary. This motivates the ASIC-oriented methodology adopted in this thesis, where model design (quantization, topology, buffering) and hardware architecture (datapaths, memory organization, control) are co-optimized to satisfy ISC-grade constraints.

I.2.3 TinyML Development Flow and ASIC-Oriented Extensions

Bringing all the aforementioned techniques together, practitioners have converged on a standard TinyML development workflow that starts from a high-level model design and ends with deployment on an embedded target. At a high level, the process involves:

- training or selecting a compact, efficient model architecture with the task's requirements in mind,

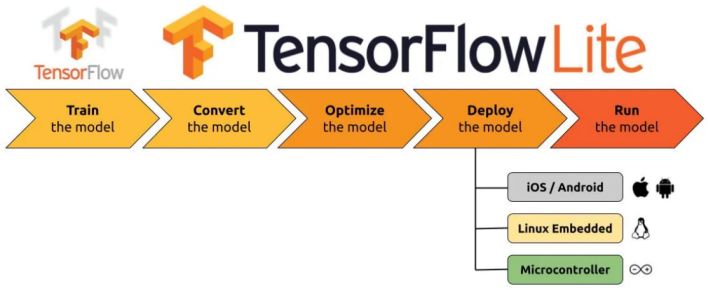

Figure I.5: *Workflow of TensorFlow Lite for TinyML applications, illustrating the steps of training, converting, optimizing, deploying, and running a model across platforms such as iOS/Android, embedded Linux, and microcontrollers. Image extracted from (Cavagnis, 2021).*

- applying compression and optimization (quantization, pruning, etc.) to the model,
- validating the model's performance and resource usage via simulation or on-device testing,
- deploying the model onto the hardware and iterating as necessary.

Importantly, this flow is usually iterative and *hardware-aware* from the start. For example, during training, one might already incorporate quantization awareness (e.g. using fake-quantization operations in TensorFlow) so that the model will not degrade significantly after being converted to integer arithmetic (Jacob et al., 2018).

Similarly, architecture decisions are guided by profiling: a developer may estimate the MAC operations, peak memory footprint, and energy per inference of a candidate model early on, using tools or analytical models, and compare these against the device's limits.

There are now automated tools to assist in this stage; for instance, Google's `model_optimizer` and frameworks like *Edge Impulse* can compute the RAM/Flash usage of a model and even simulate latency on a given MCU. The MLPerf Tiny benchmark suite has also provided a helpful reference by defining representative TinyML tasks and metrics (accuracy, latency, energy) for fair evaluation (Reddi et al., 2021). These metrics serve as design targets during development – for instance, an always-on keyword spotting model might be designed to run under 20 ms and 1 mJ per inference on an Arm Cortex-M4.

Once a model is trained and compressed to a suitable form, the next step is deploying and testing it on the actual hardware (or a development board representative of it). Frameworks like *TensorFlow Lite Micro*, whose workflow is illustrated in Fig. I.5, greatly simplify this step by handling the low-level details of inference on microcontrollers (David et al., 2021). The developer converts the model (often a TensorFlow or ONNX model) into a quantized FlatBuffer format that TFLite Micro can load. The framework then uses a kernel library (such as CMSIS-NN or its own optimized kernels) to execute the model. At compile time, memory planners in TFLite Micro determine a static memory allocation for all intermediate tensors to ensure the model fits in the available SRAM. Alternatively, vendor-provided tools like STMicro's

STM32Cube.AI can take a trained model and generate C code (or even assembly) that implements the neural network, along with an API to run inferences on that model. These tools often provide reports on memory usage and runtime cycles. By flashing the model onto a device and running inferences (either on real data or synthetic tests), one can measure actual latency (with timers) and energy (with power monitors or estimations) to verify that the deployment meets requirements. If any metrics are off – for example, if the model uses too much RAM or runs too slowly – the developer must return to the design stage and apply further optimizations: this could mean pruning additional neurons, reducing model dimensionality, or even collecting more data to allow a smaller model to achieve the required accuracy. In practice, multiple iterations are common before an optimal balance is found. Each iteration might involve adjusting the neural network (architecture or training hyperparameters) and/or making better use of hardware capabilities (such as choosing a faster numeric format or leveraging a different library).

This standard TinyML workflow has proven effective for deployment on off-the-shelf microcontrollers and SoC accelerators, and it also lays the groundwork for custom hardware development. In fact, the methodology proposed in this thesis is directly inspired by the TinyML model-deployment co-design loop.

We extend the flow one step further: rather than stopping at running the model on existing hardware platform, we translate the optimized model into a custom **application - specific integrated circuit (ASIC)** implementation. The process begins in the same fashion – with data preparation and model design, including quantization and compression to fit a micro-scale power and area budget. We verify the network on a microcontroller or simulator to ensure its feasibility. Then, in a new stage unique to ASIC development, we co-design a specialized hardware architecture around this model, leveraging the fixed nature of the neural network to customize memory structures, datapaths, and control for maximal efficiency. The final steps involve hardware description (RTL design), verification against the quantized model, and physical implementation (synthesis, place-and-route) as will be detailed in Chapter 2.

Crucially, throughout this process we maintain the iterative spirit of the TinyML flow: if the post-layout hardware analysis indicates that power or area targets are not met, we loop back to adjust the model (or the hardware parameters) and try again. This co-design approach ensures that both the neural network and the silicon meet in the middle to satisfy the extreme constraints of edge AI. Thus, the state-of-the-art techniques reviewed in this section – from MobileNet-style architectures and quantization to CMSIS-NN and accelerator design principles – collectively inform the custom low-power neural network accelerator presented later in this thesis.

In summary, deploying neural networks at the edge is a multidisciplinary endeavor, and the best results come from simultaneously tailoring the algorithm *and* the hardware. The following chapters will build upon this foundation to develop an ultra-low-power neural network processing unit, demonstrating how far TinyML can be pushed.

Chapter 2
Methodology

Designing neural network accelerators for ISC closely aligns with core principles of TinyML, which targets ML deployment on resource-constrained embedded platforms. TinyML's success hinges on an integrated hardware–software co-design process, due to the stringent energy, memory, computation, and latency requirements of embedded devices (Lin et al., 2023).

Our methodology extends these principles to the even more demanding domain of ISC, where computation must occur within the sensor itself rather than in an external microcontroller.

This chapter presents the adopted methodology, adapted from the TinyML design philosophy, with a focus on software–hardware co-design to meet the extreme requirements of in-sensor environments.

II.1 Overview of the Co-Design Flow

As illustrated in Fig. II.1, the proposed methodology follows a *constraints-first* software–hardware co-design approach tailored to ISC. Before entering the iterative design loop, the flow starts with a **Requirements and Specifications Definition** stage, which is essential to propagate application- and sensor-level constraints across all subsequent phases.

The complete methodology is therefore structured into the following stages:

1. **Requirements and Specifications Definition**
2. **Data Preparation**
3. **Neural Network Modeling**
4. **Hardware Design**
5. **Hardware Implementation**

The initial requirements-definition stage establishes the quantitative and qualitative constraints that drive the entire design process. These include target minimum accuracy or residual error, maximum allowable latency, average and peak power budgets compatible with the target operation, the silicon area and memory limits imposed by sensor integration, and assumptions on sensing modality, data rate, and interfaces.

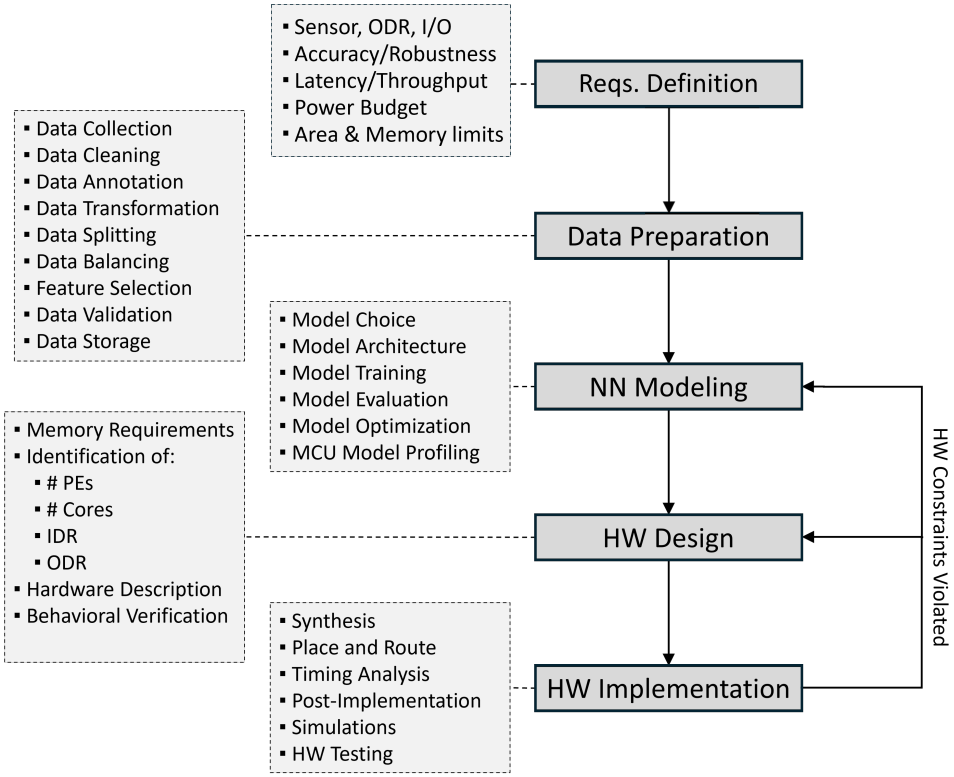

Figure II.1: *Software–Hardware co-design methodology for in-sensor neural network accelerators. The flow starts with a requirements and specifications definition stage, followed by Data Preparation, Neural Network Modeling, Hardware Design, and Hardware Implementation. An iterative feedback loop is included: if hardware constraints are not satisfied, the flow returns upstream to refine the neural model or, when necessary, the initial specifications.*

By explicitly defining these requirements upfront, infeasible design paths can be discarded early, and late-stage redesign iterations can be significantly reduced.

The remaining stages are interconnected through an iterative software–hardware co-design loop. Constraints emerging from hardware design or implementation (e.g., area overruns, excessive power consumption, or timing violations) are fed back upstream, potentially triggering revisions in the neural network architecture or, if necessary, a refinement of the initial specifications. This iterative feedback mechanism is essential for balancing accuracy, resource usage, and deployability within the severe constraints of ISC environments.

For clarity in interpreting Fig. II.1, the dotted-border blocks associated with each stage represent *non-exhaustive checklists* of typical tasks that can be performed within that stage. They are not intended to encode task priorities; instead, the relative importance and priority of activities is dictated by the application requirements and by the iterative feedback arising during the co-design process.

II.2 Requirements and Specifications Definition

The first stage of the proposed methodology formalizes the *requirements and specifications* that must be satisfied by the overall ISC system. This step is particularly critical in in-sensor computing, where the design space is strongly bounded by sensor-integration constraints (silicon footprint, limited on-chip memory, and tight power envelopes), and where late-stage changes can be costly due to the strong coupling between algorithmic choices and hardware implementation.

In this thesis, requirements are defined at two complementary levels:

- *application-level requirements*, which describe the expected functional performance and real-time behavior,
- *implementation-level constraints*, which describe the physical feasibility of integration within (or next to) the sensing element.

The outcome of this stage is a compact set of quantitative targets and design assumptions used as inputs to the subsequent phases of Fig. II.1.

The following requirements are considered as the minimal set driving the co-design process:

- **Functional performance targets:** required accuracy (classification/regression), maximum tolerable residual error, and robustness metrics (e.g., false-alarm/false-negative rate when applicable), together with a clear validation protocol.
- **Real-time constraints:** maximum allowable end-to-end latency and/or minimum throughput. Depending on the sensing modality, this can be expressed as a maximum inference time per window/frame or as a maximum update period.
- **Power envelope and operating mode:** average and peak power budgets compatible with the target operation. This includes the intended duty-cycling strategy (always-on vs. event-driven operation), which is often a first-order requirement in ISC.
- **Area and memory budgets for sensor integration:** maximum silicon footprint and maximum on-chip memory (for weights and intermediate activations), consistent with sensor packaging and integration constraints.
- **Sensing and I/O constraints:** signal modality, sampling rate / output data rate (ODR), resolution/dynamic range, and available interfaces (e.g., SPI/I^2C/PDM), which bound input bandwidth, buffering, and data representation.

The requirements above are not independent: they jointly define a feasible region in the design space and shape decisions throughout the flow. In particular, **functional targets** (accuracy, residual error, robustness) directly affect *Data Preparation*, since they determine dataset coverage, labeling policies, and the evaluation protocol used to validate the solution; they also constrain *Neural Network Modeling*, by bounding the admissible model families, training regimes, and the extent of quantization/compression that can be tolerated without violating performance objectives. These targets further propagate to *Hardware Design*, because operator support and

numerical formats must ultimately preserve the required accuracy under the selected precision.

Similarly, **real-time constraints** (latency and/or throughput) influence early choices on framing and buffering during *Data Preparation* (e.g., window length, overlap, and update rate), and they limit the complexity of the candidate neural architectures during *Neural Network Modeling* (depth/width, activation footprint, and scheduling assumptions). At the hardware level, the same constraints drive architectural decisions in *Hardware Design* and *Hardware Implementation*, such as the degree of parallelism, the execution schedule, and the target clocking strategy required to meet deadlines.

The **power envelope and operating mode** (always-on vs. event-driven) is a first-order driver in ISC. It encourages early feature representations and preprocessing choices that reduce data movement and redundant computations in *Data Preparation*. It also motivates resource-aware modeling decisions (e.g., low-bit quantization, compression, and architectures with limited intermediate activations) in *Neural Network Modeling*. Finally, it strongly impacts *Hardware Design* and *Hardware Implementation*, where clock/power gating, resource reuse, and leakage-aware implementation strategies become essential to achieve ultra-low *average* power.

In parallel, **area and on-chip memory budgets** impose hard bounds on the feasibility of sensor integration. They constrain the maximum parameter count and, critically, the intermediate activation sizes during *Neural Network Modeling*, often forcing compact topologies and aggressive precision reduction. They also dominate *Hardware Design* by limiting the number of processing elements, local buffers, and the complexity of the control logic; ultimately, they influence *Hardware Implementation* through technology/library choices and physical closure.

Finally, **sensing and I/O constraints** (signal modality, ODR, resolution, and interface bandwidth) shape the input representation and preprocessing complexity in *Data Preparation*, and they influence the input dimensionality and data layout assumptions adopted during *Neural Network Modeling*. At the architectural level, these constraints bound streaming datapaths and buffering requirements in *Hardware Design*, since the accelerator must sustain the sensor data rate while respecting tight local-memory limits.

II.3 Data Preparation

As in most machine learning pipelines—including those in TinyML—the first phase involves careful preparation of data (Mohammed et al., 2025; Whang et al., 2023). For ISC, however, the sensor type and signal characteristics strongly influence the adopted strategy, since the raw signals are acquired under strict constraints of memory, energy, and bandwidth. Data preparation is therefore not only a prerequisite for model training, but also a fundamental step to guarantee deployability in ultra-constrained environments (Samanta et al., 2024; Lin et al., 2023).

- **Data Collection**: Define the task and success metrics, then acquire data from trustworthy sources (sensors, logs, public datasets, APIs) with proper consent/licensing. Record rich metadata (timestamps, device/user/context) to support later filtering and audits.
- **Data Cleaning**: Enforce a schema; remove duplicates, fix corrupt records, re-

solve inconsistent units/encodings, and handle missing values (impute or drop). Identify and cap/outlier values; normalize formats (e.g., UTF-8, UTC).

- **Data Annotation**: Create clear labeling guidelines and train annotators; capture labels, confidence, and rationales.
- **Data Transformation**: Convert raw inputs into model-ready representations (scaling/standardization, tokenization, one-hot/ordinal encoding, feature engineering). Apply domain-appropriate augmentation while preventing target leakage.
- **Data Splitting**: Partition into train/validation/test. Consider cross-validation or time-based splits for robust estimation.
- **Data Balancing**: Address class imbalance with class weights, under/oversampling, or carefully designed augmentation without duplicating test data.
- **Feature Extraction and Validation**: Extract and select task-appropriate representations—aim to reduce dimensionality and improve generalization while preserving informative variance (e.g. with filtering, correlation, mutual information, Principal Component Analysis (PCA) , Independent Component Analysis (ICA), Nonnegative Matrix Factorization (NMF), autoencoders (AEs)). Validate choices with ablations and cross-validation, and freeze the final feature spec before testing.
- **Data Storage**: Store datasets in versioned, immutable formats (e.g., Parquet/HDF5). Include metadata (schema, units, licenses), documentation, and enforce governance: access control, PII handling, retention, and reproducibility.

These processes were carried out using MATLAB and Python (NumPy, Pandas, SciPy), with STM32CubeIDE employed for preliminary sensor interfacing. Particular attention was paid to ensuring that preprocessing steps remained compatible with the target hardware, avoiding complex transformations that would exceed the available energy or memory budgets.

II.4 Neural Network Modeling

Once the data have been properly prepared, the next stage focuses on designing neural network models that can operate within the extreme constraints of in-sensor environments. Rather than pursuing maximum accuracy in isolation, the modeling process is hardware-aware from the outset, with a multi-objective target that balances predictive performance, memory footprint, latency, and energy. In practice, early profiling and device-specific constraints (e.g., on-chip SRAM and allowable activation sizes) guide architectural choices and prevent designs that would later prove infeasible during implementation (Lin et al., 2020b; Banbury et al., 2021).

- **Model Selection**: We favour compact families (e.g., shallow CNNs, depthwise-separable CNNs, lightweight autoencoders) and, for extreme budgets, binary/ternary networks. The choice is tied to the signal modality and the feature representation produced in the previous stage (e.g., 2D time–frequency maps vs.

raw sequences). Hardware-aware search methods (e.g., TinyNAS) and curated TinyML spaces (e.g., MicroNets) help identify architectures that respect memory/latency constraints on microcontrollers, where operation count correlates strongly with measured latency (Lin et al., 2020b; Banbury et al., 2021).

- **Architecture Design**: Depth, width, kernel sizes/strides, and activation functions are tailored to minimize intermediate activation footprints and off-chip transfers. Structural patterns such as inverted residual blocks and depthwise separable convolutions reduce MACs substantially while preserving accuracy in embedded regimes. For ultra-constrained targets, binarized and ternary weights can slash model size and replace multiply-accumulate with bit-wise operations (Yizhi Wang et al., 2018; J. Woo et al., 2024).

- **Training and Evaluation**: Training is performed offline with standard optimizers and data augmentation; validation uses representative distributions compatible with deployment. Quantization-friendly training (e.g., fake-quant nodes, per-channel scales) is adopted early to reduce post-training surprises. When available, hardware-in-the-loop or cycle-accurate simulators provide early timing checks (Jacob et al., 2018).

- **Model Optimization**: We apply a toolkit of techniques—post-training quantization (PTQ), quantization-aware training (QAT), pruning, and knowledge distillation—to meet tight memory/energy budgets with limited accuracy loss (Zihan Wang et al., 2022; Han et al., 2015a; Jacob et al., 2018). In addition, hardware-friendly implementations of critical operators are evaluated early, including polynomial or piecewise approximations for activation functions, saturation/rounding policies, and fixed-point scaling strategies. Such approximations are particularly relevant for non-linear operators (e.g., $\tanh(\cdot)$) and are retained only if their induced numerical error is bounded below the chosen quantization step (i.e., it does not affect the effective output resolution); the selected implementation is then carried forward consistently into the RTL design.

- **MCU Model Profiling and Deployment Estimation**: Before translating the model to custom hardware, we perform an early feasibility assessment on a representative MCU-class target. This step combines analytical and toolchain-based estimates to characterize computational load (MACs/ops), parameter and activation footprints, and peak SRAM usage, together with early latency and energy proxies.

As in the rest of the methodology, this stage is inherently iterative: when the toolchain feedback reveals violations (e.g., SRAM overflow, timing margins, or energy per inference), the model is revised and retrained until the design satisfies both algorithmic and hardware constraints.

All modeling experiments were carried out using Python-based frameworks, primarily TensorFlow, Keras, and QKeras for training, validation, and testing. MCU profiling and deployment checks relied on TensorFlow Lite Micro and STM32Cube.AI, providing a direct link between algorithmic choices and hardware feasibility (Martín Abadi et al., 2015; Coelho et al., 2021).

II.5 Hardware Design

Once the neural network has been selected, optimised with hardware awareness, and profiled on a microcontroller, the next step is to map it onto a custom accelerator that can satisfy the stringent constraints of in-sensor environments. Unlike general-purpose TinyML toolchains targeting microcontrollers, here the design is tailored to custom ASICs or FPGAs to maximise performance and energy efficiency within the extreme constraints of in-sensor environments. The design process prioritises data reuse, controlled memory traffic, and predictable latency under small memory and power budgets, while keeping the micro-architecture simple enough for robust verification and physical implementation.

Key steps include:

- **Memory Analysis**: Early estimation of buffer sizes, weight storage, and bandwidth requirements, together with selection of numerical formats. Fixed-point quantisation (typically 8-bit or lower) is adopted to reduce both memory and MAC cost, while accumulation widths are carefully dimensioned to avoid overflow.
- **Hardware Architecture Definition**: Definition of the compute core, typically an array of lightweight PEs with local buffers and MAC units, along with hardware blocks for activation functions and pooling. The goal is to maximise data reuse (weights, inputs, and partial sums) and minimise off-chip memory transfers, since data movement dominates energy. A lightweight controller coordinates layer execution and data flow. Techniques such as clock gating and separate clock domains are adopted to minimise dynamic power and ensure reliable always-on operation. Metrics are expressed not only in area and frequency, but also in energy per inference (nJ/inf), which is the most relevant KPI in ISC scenarios.
- **Hardware Description**: RTL design using VHDL/Verilog hardware description language.
- **Behavioral Verification**: Functional correctness is validated against the quantised reference model simulating the RTL design using directed and constrained-random testbenches.

II.6 Hardware Implementation

The final phase validates the co-designed model via prototyping and physical implementation. This step is structured in following phases:

- **FPGA Prototyping**: The RTL is mapped to Xilinx FPGAs using Vivado Design suite to test timing, latency, throughput, and functionality under realistic streaming I/O conditions. This step provides early feedback on performance and buffering before committing to an ASIC flow.
- **ASIC Implementation**: The validated RTL is mapped to a standard-cell CMOS process (e.g., 130 nm/65 nm). This phase includes:

- Logic synthesis (Synopsys Design Compiler) with MCMM constraints to ensure timing and area feasibility.
- Place-and-Route (Cadence Innovus) with congestion/timing closure and clock/power domain management.
- Power and Timing Analysis (PrimeTime, Voltus) with vector-based estimation of dynamic and leakage power, expressed as energy per inference.
- Post-layout Simulations (gate-level with SDF back-annotation) and equivalence checking against the quantised software model.
- Physical Testing when fabrication is available.

These steps allow accurate estimation of area, maximum frequency, and energy per inference. Unlike general-purpose TinyML targets, ASIC synthesis enables direct integration within sensor packages while optimising for always-on operation under strict power budgets.

If hardware synthesis reports violations—such as excessive power, area, or timing—the flow returns to the hardware design stage. In this case, the architecture is revisited: for instance, by increasing the reuse of a single processing module (thus reducing the number of instantiated modules and saving area at the cost of latency), or conversely by instantiating additional modules to accelerate computation at the expense of area.

If these architectural refinements are still insufficient to meet constraints, the process falls back to the neural network modeling phase, where the model is simplified or further optimised.

The flow then restarts iteratively until both functional and physical goals are achieved.

The defining feature of this methodology is precisely this continuous integration between algorithmic design and hardware feedback. The co-design loop ensures that models are optimised not only for accuracy, but also for real-world deployability under the stringent requirements of in-sensor computing.

This approach, while inspired by TinyML practices, is adapted to the even more constrained ISC landscape, enabling the creation of neural accelerators that are simultaneously efficient, physically realisable, and suitable for integration directly within sensor packages—unlocking a new generation of smarter and more autonomous sensor systems.

Chapter 3
Case 1: Neural Network-Based In-Sensor PDM-to-PCM Conversion for Ultra Low-Power TinyML Keyword Spotting

III.1 Publications and Awards Associated with This Chapter

The case study presented in this chapter builds upon peer–reviewed publications produced during the PhD's broader research on KWS systems, conducted in collaboration with STMicroelectronics.

This research resulted in two Q1-journal papers, three conference proceedings and one best presented paper award. Readers may refer to the following for further details.

III.1.1 Journal Articles

- Paola Vitolo et al. (2023c). "A New NN-Based Approach to In-Sensor PDM-to-PCM Conversion for Ultra TinyML KWS". in: *IEEE Transactions on Circuits and Systems II: Express Briefs* 70.4, pp. 1595–1599. DOI: 10.1109/TCSII.2022.3224022

- Paola Vitolo et al. (2024c). "Automatic Audio Feature Extraction for Keyword Spotting". In: *IEEE Signal Processing Letters* 31, pp. 161–165. DOI: 10.1109/LSP.2023.3346280

III.1.2 Conference Proceedings

- Paola Vitolo et al. (2022b). "Quantized ID-CNN for a Low-power PDM-to-PCM Conversion in TinyML KWS Applications". In: *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 154–157. DOI: 10.1109/AICAS54282.2022.9869909

- Paola Vitolo et al. (2023b). "A 0.8 mW TinyML-Based PDM-to-PCM Conversion for In-Sensor KWS Applications". In: *Proceedings of SIE 2022*. Ed. by

Giuseppe Cocorullo et al. Cham: Springer Nature Switzerland, pp. 146–151. ISBN: 978-3-031-26066-7. DOI: 10.1007/978-3-031-26066-7_23

- Paola Vitolo et al. (2023a). "In-sensor neural network for real-time KWS by image processing". In: *Real-time Processing of Image, Depth and Video Information 2023*. Vol. 12571. International Society for Optics and Photonics. SPIE. DOI: 10.1117/12.2665545

III.1.3 Awards

- *AICAS 2022 LG Electronics Award* for the paper "Quantized 1D–CNN for a Low–power PDM-to-PCM Conversion in TinyML KWS Applications," presented at the *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (Vitolo et al., 2022b)*.

III.2 Introduction

Over the past decade, voice interaction has become an increasingly widespread modality for human–machine communication. Once limited to niche applications, voice user interfaces are now commonly embedded in smart speakers, mobile phones, domestic appliances, vehicles, industrial settings and even healthcare and assistive devices (Deshmukh and Chalmeta, 2024; Papavasileiou et al., 2024; Bramanti et al., 2025; Hamidi et al., 2020). Voice assistants enable hands-free control, accessibility, and safer human–machine interaction, for example, helping drivers keep their eyes on the road, allowing elderly users to operate home devices, and enabling contactless/sterile interaction in clinical contexts (Vincze et al., 2025; Ambewadikar and Baheti, 2020; Sindhu and Sainin, 2024).

For edge voice assistants, neither a purely on-device nor a purely cloud-based Automatic Speech Recognition (ASR) architecture is satisfactory. On microcontroller-class hardware, running large-vocabulary ASR continuously is out of reach: TinyML deployments operate under tight memory (often < 1 MB flash), compute, and energy budgets, so full ASR would violate latency and power envelopes (Heydari and Mahmoud, 2025; Alajlan and Ibrahim, 2022).

Conversely, cloud-only ASR introduces network-dependent delay and jitter, increases energy due to continuous radio transmission, and raises bandwidth and privacy concerns when streaming raw audio off the device (Bolton et al., 2021; Heydari and Mahmoud, 2025; Solera-Cotanilla et al., 2022). In practice, an always-on, small-footprint KWS block acts as a power- and compute-gating stage: it continuously listens for a short wake word and activates the heavier ASR only on demand—the reference design adopted in voice assistants (G. Chen et al., 2014; Heydari and Mahmoud, 2025).

Therefore, as illustrated in Fig. III.1, a typical edge voice-assistant pipeline comprises a sensing element (microphone), a front-end for signal conversion and preprocessing, an always-on KWS module that detects a wake word, and—upon detection—a larger ASR stage executed on a more capable processor in the cloud (G. Chen et al., 2014; Cámbara et al., 2022).

In this context, digital MEMS microphones play a central role as front-end audio

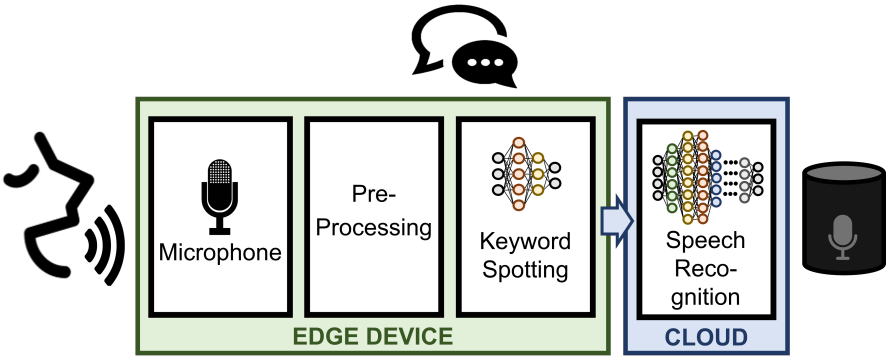

Figure III.1: *Typical edge voice-assistant pipeline. A digital microphone captures audio and feeds a front-end for conversion and preprocessing; an always-on keyword-spotting (KWS) module listens for the wake word and, upon detection, activates the more compute-intensive automatic speech recognition (ASR) in the cloud.*

sensors. Figure III.2 shows the inside of a typical digital package, where the MEMS transducer is co-packaged with a readout ASIC; this monolithic integration reduces analog routing, improves robustness, and eases board-level integration. At the same time, thanks to small form factor and manufacturing scalability, MEMS microphones deliver low cost and straightforward embedding in edge devices (STMicroelectronics, 2017; Zheng et al., 2024).

Compared with analog alternatives, the digital output is less susceptible to RF/EMI picked up along PCB traces—an advantage in electrically noisy edge platforms—and recent designs demonstrate low-power, compact readouts suitable for battery-operated devices (F. Li et al., 2023; T. Wang et al., 2025).

As summarized in the block diagram of Fig. III.3, the MEMS transducer feeds a low-noise amplifier and an ADC; a $\Sigma\Delta$ PDM modulator then generates an oversampled 1-bit data stream, while on-chip power management and a channel-select pin provide biasing and L/R selection. The standard digital interface exposes *CLK* and

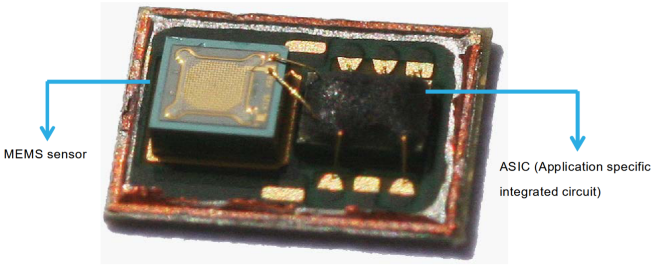

Figure III.2: *Inside a digital MEMS microphone package: the MEMS transducer (left) and the signal-conditioning/readout ASIC (right). Image extracted from STMicroelectronics,* Tutorial for MEMS Microphones*, Application Note AN4426 (DocID025704 Rev. 2) (STMicroelectronics, 2017).*

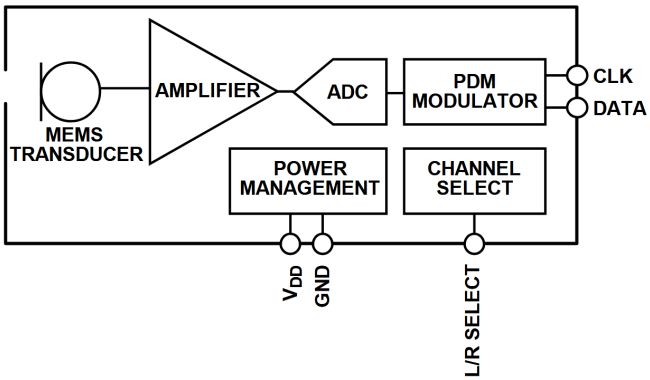

Figure III.3: *Block diagram of a digital MEMS microphone. The MEMS transducer feeds a low-noise amplifier and an ADC; a $\Sigma\Delta$ PDM modulator generates the 1-bit data stream, while power-management and channel-select circuitry provide biasing and L/R selection. Outputs are CLK and DATA. Extracted from Analog Devices,* Analog and Digital MEMS Microphone Design Considerations*, Technical Article MS-2472 (Lewis, 2013).*

DATA lines (Lewis, 2013). Because downstream audio processing and classification typically operate on multi-bit *PCM* at kHz rates, the microphone's PDM output must first be converted to PCM—making the PDM $\rightarrow$ PCM interface a mandatory stage in any edge voice-assistant pipeline.

However, traditional PDM-to-PCM conversion relies on multi-stage filtering and decimation pipelines—such as Cascaded Integrator-Comb (CIC) filters—which often dominate the power and area budget.

This chapter presents a novel approach to PDM-to-PCM conversion based on a tiny, quantized 1D Convolutional Neural Network (1D-CNN), specifically designed for in-sensor integration. The proposed method unifies filtering and decimation into a single learnable block, overcoming the limitations of conventional solutions and enabling tight coupling with downstream neural KWS classifiers. The system is fully described in hardware, prototyped on FPGA, and synthesized in 130 nm CMOS standard-cell technology.

The proposed converter achieves a signal-to-noise ratio (SNR) of 48 dB and supports 8-bit PCM outputs at 16 kHz, compatible with existing tinyML audio classifiers. Despite its neural nature, the design fits within an area of 0.086 mm^2 and consumes only 128.7 µW/MHz, making it suitable for integration within MEMS microphone packages. Additionally, it enables a complete end-to-end KWS pipeline with 89% classification accuracy over 12 audio classes.

The remainder of this chapter is organized as follows: Section III.3 reviews the state of the art in PDM-to-PCM conversion and related neural filtering techniques. Section III.4 introduces the proposed 1D-CNN-based method, including its architecture, dataset generation, and training setup. Section III.5 details the hardware implementation and synthesis results, discussing the broader implications and potential for in-sensor integration.

III.3 State of the Art

III.3.1 Traditional PDM-to-PCM Conversion Approaches

In edge audio systems based on digital MEMS microphones, raw acoustic signals are encoded as oversampled 1-bit PDM bitstreams. Before feature extraction or classification, this stream must be converted to multi-bit PCM at kHz rates.

Conventional PDM-to-PCM conversion pipelines employ complex multi-stage high-order filtering and high-value-factor decimation, which hardly fit resource constraints for in-sensor computing (F. Zhou and Chai, 2020; R. Wan et al., 2022).

CIC filters are the widely diffused solution that circumvents the problem by avoiding multipliers and memory for filter coefficients, resulting in a HW efficient implementation (Stošić, 2021; Hogenauer, 1981). However, CIC advantages are partially nullified by the additional filtering operations required to compensate for the poor cut-off and to remove the aliasing. Indeed, despite their hardware efficiency, CIC-based pipelines suffer from several drawbacks that limit their applicability in ultra-constrained environments such as in-sensor computing.

First, the sharp roll-off of the CIC filter response leads to significant in-band attenuation and distortion, especially when the filter length is minimized to save area. This degrades the quality of the recovered audio and ultimately reduces the performance of downstream classifiers.

Second, CIC filters and their compensation stages require large internal word widths to avoid overflow, especially in high-decimation configurations. This increases memory usage and routing complexity in digital hardware, which is undesirable near the sensor.

Moreover, fixed filtering kernels are unable to adapt to different audio environments or compensate for variations in sensor characteristics.

III.3.2 NN-Based Solutions in the Literature

Recent research efforts have explored the application of data-driven models, such as NNs. In particular, they have been applied to denoising, audio super-resolution, and end-to-end audio classification, often achieving superior performance compared to handcrafted solutions (Zaman et al., 2023).

However, the use of neural networks for PDM-to-PCM conversion remains largely unexplored. A few recent works have proposed leveraging the learning capability of neural networks to approximate the desired frequency response and remove quantization noise, offering the potential for more compact and adaptable designs.

(Alwahab et al., 2018) uses a neural network to initialize and refine FIR coefficients with a custom frequency-domain loss, improving passband flatness, transition steepness, or stopband attenuation. Likewise, (Koh, 2021) proposes a generative-adversarial approach that learns families of FIR filters at arbitrary cut-off frequencies from ideal time-domain templates.

However, existing NN-based approaches do not investigate the design of decimation filters, which is essential in PDM-to-PCM conversion as they determine the quality of the signals passed from digital MEMS microphones to audio processing systems. Nonetheless, most of these studies target relatively large models unsuitable for

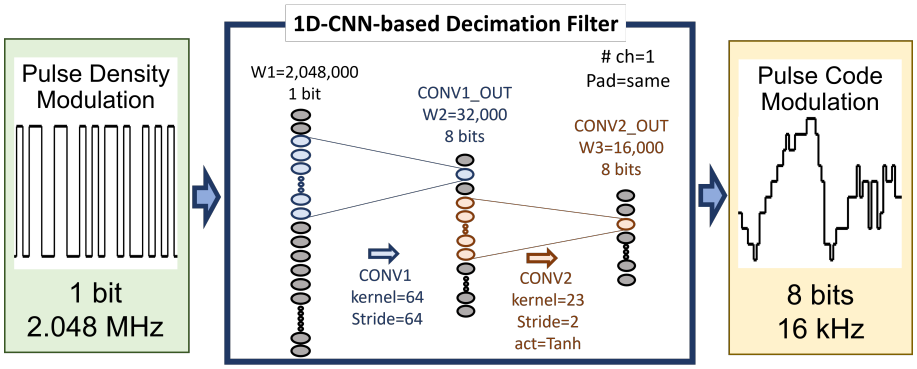

Figure III.4: *Proposed neural PDM→PCM interface. A 1-bit PDM stream at 2.048 MHz is processed by a tiny 1D-CNN that fuses filtering and decimation:* ***CONV1*** *(kernel=64, stride=64) reduces the rate to 32 kHz;* ***CONV2*** *(kernel=23, stride=2,* $\tanh$*) further decimates to 16 kHz, yielding 8-bit PCM. Overall decimation* $R = 128$*, replacing the conventional CIC+FIR chain.*

in-sensor deployment or rely on post-processing stages executed on general-purpose processors. To the best of our knowledge, there are no prior demonstrations of a fully hardware-synthesizable, ultra-low-power neural PDM-to-PCM converter suitable for direct integration within MEMS microphone packages.

III.4 The Proposed Method for the PDM-to-PCM Converter

III.4.1 1D-CNN Architecture for PDM-to-PCM

The proposed PDM→PCM conversion block diagram is depicted in Fig. III.4. It is based on a tiny one-dimensional convolutional neural network (1D-CNN) explicitly optimized for in-sensor hardware deployment. A convolutional front-end is preferred over fully connected (FC) layers because (i) *local receptive fields* and *weight sharing* drastically reduce the number of parameters and multiply–accumulate (MAC) operations compared to dense connections, (ii) strided convolutions perform rate-change natively, enabling decimation without separate blocks, and (iii) the spatial/temporal stationarity of audio signals is well captured by convolution while preserving small memory footprints—properties that have long motivated CNNs in resource-limited acoustic modeling (Lecun et al., 1998; Alzubaidi et al., 2021; Z. Li et al., 2022).

The network processes a high-rate (2.048 MHz), single-bit PDM input and emits 8-bit PCM samples at 16 kHz, directly suitable for downstream feature extraction and KWS classification. In line with the Google Speech Commands (GSC) setup, the useful audio bandwidth is limited to 0–8 kHz; typical digital MEMS microphones operate at MHz-rate PDM outputs, hence the need for large decimation to audio-rate PCM (Warden, 2018; STMicroelectronics, 2021).

The input to the model is a 1-second PDM window of $W_1{=}2{,}048{,}000$ samples (2.048 MHz). The network has two convolutional layers with one channel and *same* padding:

- **CONV1**: kernel size 64, stride 64; its output is W_2=2,048,000/64=32,000 values, quantized to 8 bits.
- **CONV2**: kernel size 23, stride 2, activation $\tanh(\cdot)$; its output is W_3=32,000/2=16,000 values, also quantized to 8 bits.

The overall decimation is therefore $R = 64 \times 2 = 128$, mapping the 2.048 MHz, 1-bit PDM stream to a 16 kHz, 8-bit PCM waveform. The hyperbolic tangent is defined as

$$\tanh(x) \;=\; \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{III.1}$$

For system-level evaluation and to verify deployability within an end-to-end KWS pipeline, the 8-bit PCM produced by the converter is fed to a compact, 8-bit quantized KWS model operating on GSC-style features, representative of tinyML classifiers used on microcontroller-class devices (MLCommons, 2021). It is an already trained TFLite model, which accepts 10×49 8-bit mel-frequency cepstrum as input and is capable of 92% accuracy over the 12 classes, trained with the Google Speech Commands Dataset (GSCD), which is a public dataset that has become the de facto open benchmark for KWS development and evaluation (Warden, 2018).

III.4.2 Hardware-aware activation approximation.

Since the second convolutional layer employs a $\tanh(\cdot)$ non-linearity, its implementation was considered during the early, hardware-aware modeling stage (as part of the co-design flow described in Chapter II). In particular, to avoid LUT-based implementations and preserve a compact datapath, we evaluated a low-order polynomial approximation of $\tanh(\cdot)$ through fixed-point, operator-level simulations. The approximation was retained only after verifying that, over the intended operating range, the induced numerical error remains bounded below the 8-bit quantization step (approximately 2^{-7}), i.e., it does not affect the effective output resolution at the layer boundary. The selected approximation is then carried forward consistently into the RTL implementation, as detailed in Sec. III.5 (Eq. (III.5)).

III.4.3 Data Preparation for the Proposed PDM-to-PCM Converter Model

Training the 1D-CNN for PDM-to-PCM conversion requires a dataset composed of aligned PDM and PCM signal pairs, where the PDM signal serves as the input and the corresponding PCM waveform acts as the target. Since no public dataset provides raw PDM signals paired with clean PCM references, a custom dataset was generated using a simulation-based approach.

As it can be seen in Fig. III.5, the PCM audio samples were first extracted from the GSCD, the open dataset commonly used in keyword spotting (KWS) tasks. The GSCD contains 105,829 labeled utterances covering 35 words; each file is a mono 16-bit PCM waveform at 16 kHz with duration up to 1 s. For this work we selected the standard 12-class subset (10 keywords, *unknown*, and *silence*). PCM utterances were amplitude-normalized to $(-0.4,\, 0.4)$ and zero-padded to 1 s when shorter.

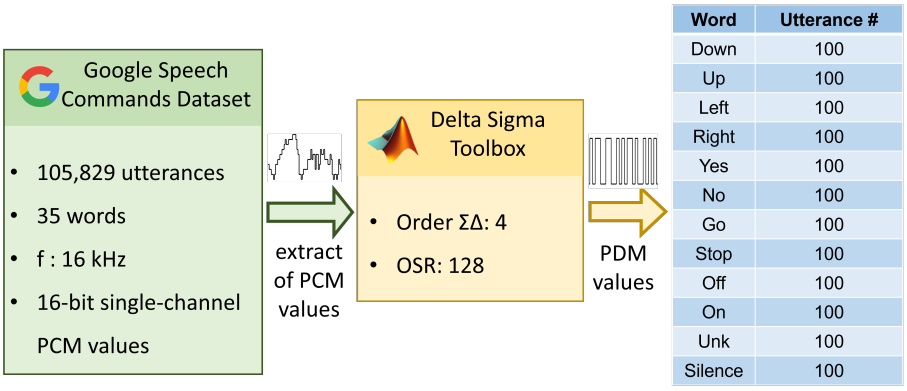

Figure III.5: *Custom PDM/PCM dataset construction. Starting from the Google Speech Commands Dataset (GSCD) (Warden, 2018), we selected 12 classes (10 keywords + unknown + silence) with 100 utterances per class. The original PCM waveforms (16 kHz, 16-bit, mono) were amplitude-normalized to* $(-0.4, 0.4)$ *and zero-padded to 1 s when shorter. Using MATLAB's Delta–Sigma Toolbox (Schreier, 2020), we generated corresponding 1-bit PDM bitstreams with a fourth-order* $\Sigma\Delta$ *modulator and an oversampling ratio of 128 (2.048 MHz). The resulting balanced dataset comprises 1,200 aligned pairs in both PCM and PDM formats.*

To synthesize realistic microphone outputs, each normalized PCM waveform was passed through a fourth-order $\Sigma\Delta$ modulator using MATLAB's Delta–Sigma Toolbox (Schreier, 2020). With an oversampling ratio (OSR) of 128, the resulting PDM bitstreams are 1-bit at 2.048 MHz, emulating the output format of digital MEMS microphones. This process yields perfectly aligned PDM/PCM pairs for supervised regression.

We constructed a balanced dataset with 100 utterances per class (1 s each), for a total of 1,200 recordings available in both PCM and PDM formats. The resulting dataset was split into 80% training, 10% validation, and 10% test sets, preserving class balance across partitions.

III.4.4 Quantization and Loss Function Design

To ensure compatibility with ultra-low-power hardware platforms and reduce memory and computation requirements, the proposed 1D-CNN model is fully quantized. Both weights and activations are quantized to 8-bit fixed-point representations, enabling efficient implementation using integer arithmetic and eliminating the need for floating-point operations.

Quantization-aware training (QAT) was employed during model development to mitigate accuracy degradation. This technique simulates quantization effects during the forward and backward passes of training, allowing the network to adapt its parameters to quantized operations.

We introduce the custom *Fast-Fourier-Transform Mean Absolute Error* (FFT–MAE) described in Eq. III.2 to approximate as much as possible the magnitude response of the desired decimation filter. The loss is the average absolute difference between the magnitude of the FFT of the model outputs y_i and the magnitude of the FFT of the

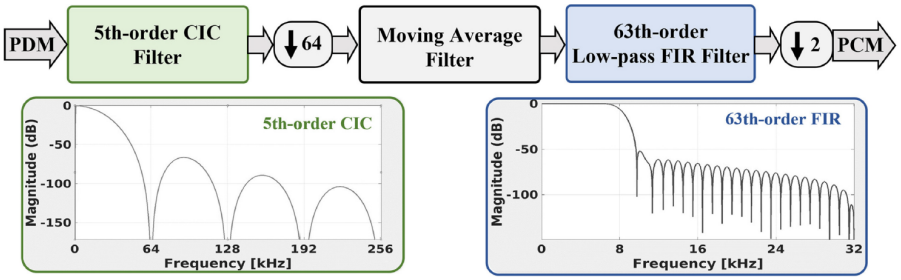

Figure III.6: *Conventional PDM→PCM multistage decimation chain. A 2.048 MHz, 1-bit PDM stream (from a fourth-order ΣΔ modulator) is first downsampled by* 64 *using a 5th-order CIC, followed by a moving-average equalizer and a 63rd-order low-pass FIR with an additional factor-2 decimation, yielding 16 kHz PCM. Magnitude responses of the CIC and FIR stages are shown. Extracted from (Vitolo et al., 2022b).*

corresponding labels $\hat{y}_i$.

$$\mathcal{L}_{\text{FFT-MAE}} = \frac{1}{n}\sum_{i=1}^{n}\Big|\,\big|\mathcal{FFT}\{y_i\}[k]\big| - \big|\mathcal{FFT}\{\hat{y}_i\}[k]\big|\,\Big| \tag{III.2}$$

III.4.5 Training and Model Evaluation

The model was trained using the Adam optimizer with an initial learning rate of 10^{-2} and batch size of 64. The training process was carried out for 150 epochs with early stopping based on validation loss to prevent overfitting.

The proposed network has been modeled and trained using TF and QKeras frameworks. The model has been initially described and trained using TF on a GPU-equipped workstation. Subsequently, the TF model has been 8-bit quantized by using QKeras and this has been fine-tuned.

To evaluate the quality of the PDM-to-PCM conversion, the following performance metrics were computed on the test set:

- **Mean Absolute Error (MAE):** Measures the average absolute difference between predicted and ground truth PCM samples.
- **FFT-MAE:** Measures the average absolute difference between the magnitude of the FFT of the model outputs and the magnitude of the FFT of the corresponding labels.
- **Signal-to-Noise Ratio (SNR):** Indicates the amount of residual noise introduced by the conversion. It is defined as:

  $$\text{SNR} = 10\log_{10}\left(\frac{\sum_i y_i^2}{\sum_i (y_i - \hat{y}_i)^2}\right)$$

  Higher SNR values indicate better fidelity.

Table III.1: *Memory for parameters and number of operators per window required by the proposed system and the CIC-based filter. Extracted from (Vitolo et al., 2023c).*

	# of ADDs	# of MULTs	Param. [Bytes]
CIC-based Filter	4,064,000	2,016,000	252
Proposed System	4,544,000	368,000	89

In addition to regression metrics, the effectiveness of the converter was also assessed in the context of a downstream keyword spotting (KWS) task. The converted PCM outputs were fed into a pre-trained neural KWS classifier operating on 16 kHz audio. The classification accuracy was used as an application-level measure of signal usability.

Results showed that FFT-MAE was 0.16 and MAE was 0.005 on the test dataset. The SNR achieved with a 1 kHz sinusoidal input is 48 dB, which is only 4% lower than the theoretical maximum SNR with 8-bit quantization. The use of FFT-MAE as loss function in place of MAE has improved the SNR of 4.3%.

When driven by the 8-bit/16 kHz PCM reconstructed by the proposed PDM-to-PCM converter, the MLCommons KWS model introduced as reference in Sec. III.4.1 achieves an end-to-end classification accuracy of 89% over the same 12 classes, corresponding to an absolute reduction of approximately 3 percentage points with respect to the nominal 92% accuracy reported for the reference pipeline. This indicates that the learned neural conversion largely preserves the discriminative audio information required for downstream keyword-spotting tasks.

In the context of this case study, this degradation is acceptable and consistent with the intended objective, namely to demonstrate that a sensor-integrable, ultra-low-power front-end can deliver a PCM stream compatible with downstream TinyML KWS inference: although the end-to-end accuracy is reduced by approximately 3 percentage points, it enables a highly optimized hardware implementation—in terms of silicon footprint and power consumption—that meets stringent ISC constraints (16 kHz, 8-bit PCM), as discussed in the following sections.

For comparison, we implemented in Matlab a conventional multistage decimation filter to serve as a reference baseline. Following established CIC design guidelines (Da Silva et al., 2019), the chain comprises a 5th-order CIC with decimation by 64, a moving-average equalizer to correct passband droop, and a 63rd-order low-pass FIR that performs an additional factor-2 decimation (overall $R{=}128$). The architecture and the stage magnitude responses are shown in Fig. III.6.

Table II compares this baseline with the proposed neural converter in terms of parameter memory and arithmetic operators. Although our design uses slightly more adders than the CIC-based chain, it reduces the number of multipliers and the parameter memory footprint by approximately one order of magnitude, yielding a markedly lower overall computational complexity and hardware resource usage. These savings, together with the achieved SNR , MAE, and FFT-MAE, make the proposed filter a practical interface for always-on KWS pipelines under in-sensor computing constraints.

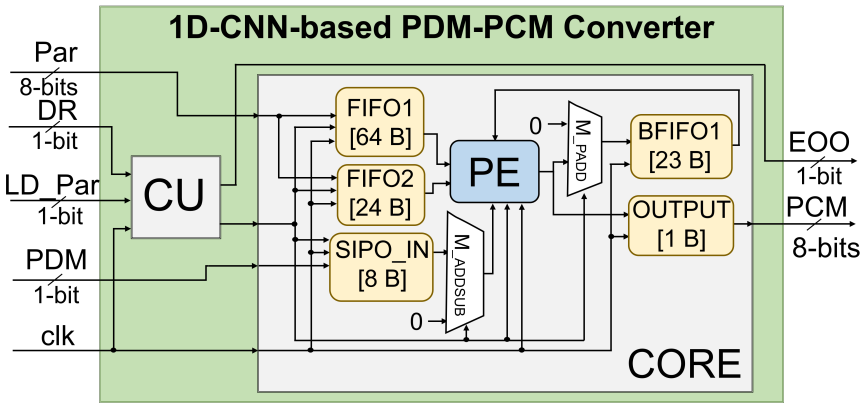

Figure III.7: *Processing Element (PE) datapath. A single 21-bit pipeline combines: an input multiplexer for operand/sign selection, a multiplier (X) used for CONV2 MACs, an adder/subtractor, and the 21-bit output register (`REG_PE`). For CONV1 the multiplier is bypassed and signed add/sub implements the $\{-1, +1\}$–weighted accumulation; for CONV2 the PE performs 24 MACs per output sample. Bit-widths are chosen to avoid overflow while operating with 8-bit quantized data.*

III.5 Proposed PDM-to-PCM Converter Hardware Implementation

III.5.1 Architecture Design of the Proposed PDM-to-PCM Converter

To meet the stringent area and power budgets of ISC—namely, to remain within a *sub-mW* power envelope and a *sub-0.1 mm*^{2} silicon footprint compatible with tight co-integration in a digital MEMS microphone readout ASIC operating at MHz-rate ODRs—we set design targets below 1 mW and below 0.1 mm^2 at an ODR of 2.048 MHz (OSR=128).

To achieve these targets, the proposed hardware adopts an *iterative* microarchitecture that maximizes resource reuse by time-multiplexing a single compute datapath across all network operations.

This minimizes memory requirements and logic replication at the cost of additional cycles—an acceptable trade-off here because the processing rate is ultimately bounded by the sensor *Output Data Rate* (ODR). In other words, real-time is ensured as long as one input sample can be fully processed before the next one arrives.

As illustrated in Fig. III.7, the accelerator consists of a *Control Unit* (CU) and a *processing core* (CORE). The CU is implemented as a finite-state machine (FSM) that sequences layer operations, configures addressing modes, and arbitrates memory ports. The CORE contains one *Processing Element* (PE)—the sole arithmetic datapath—augmented with a small set of local buffers:

- **SIPO_IN** (8 bytes): serial-in/parallel-out buffer for the incoming PDM; acts as the input FIFO.

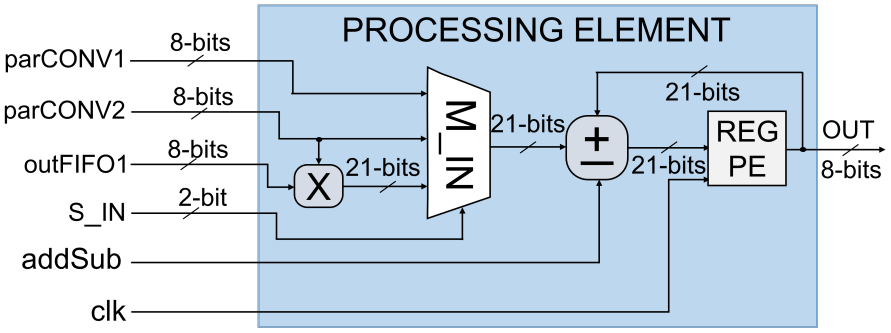

Figure III.8: *Top-level hardware of the 1D-CNN–based PDM→PCM converter. A finite-state **Control Unit** (CU) orchestrates a reusable **CORE** containing one **PE** and small byte-sized memories: `FIFO1` [64 B] (CONV1 weights/bias), `FIFO2` [24 B] (CONV2 weights/bias), `SIPO_IN` [8 B] (input PDM buffer), `BFIFO1` [23 B] (partial sums, shift/circular modes), and `OUTPUT` [1 B] (8-bit PCM). Multiplexers `M_ADDSUB` and `M_PADD` implement signed add/sub and zero-padding. The design reuses the single PE across layers to minimize area and power.*

- **BFIFO1** (23 bytes): buffer for partial sums; behaves as a *shift register* when written and as a *circular buffer* when read.
- **FIFO1** (65 bytes): parameter store for CONV1 (weights and bias).
- **FIFO2** (24 bytes): parameter store for CONV2 (weights and bias).

At start-up, the CU preloads CONV1/CONV2 parameters into `FIFO1`/`FIFO2` and sets all FIFOs to circular mode for the remainder of the run.

The PE, shown in Fig. III.8, comprises a multiplier, an adder/accumulator, a small bank of registers and a set of multiplexers for flexible routing:

- `REG_outPE`: holds the current PE output (partial sum or layer output).
- `REG_act_func`: scratch register used by the activation (tanh) engine.
- Control signals select among data sources/sinks (e.g., SIPO_IN, BFIFO1, FIFO1/2) and implement padding by injecting zeros.

Let $w_1[i]$ be the CONV1 weights and $\mathrm{pdm}[i] \in \{0,1\}$ the input bit. Mapping $\mathrm{pdm}[i]$ to $\{-1,+1\}$ by $\tilde{p}[i] = (-1)^{\neg \mathrm{pdm}[i]}$, the CONV1 output is

$$\mathrm{conv1\_out} = \sum_{i=0}^{63} \tilde{p}[i]\; w_1[i], \tag{III.3}$$

so each term is a signed add or subtract of $w_1[i]$ and no explicit multiplication is required. The CU therefore routes $w_1[i]$ to the adder with the appropriate sign based on the input bit.

CONV2 performs a standard 1-D convolution over the decimated stream:

$$\mathrm{conv2\_out}[j] = \sum_{i=0}^{22} \mathrm{conv1\_out}[j-i]\, w_2[i], \tag{III.4}$$

requiring 24 multiply–accumulate (MAC) operations per output sample. When padding is needed, the CU forces the PE input to zero via the multiplexer (i.e., *zero-insertion* at the boundaries).

As validated during the early hardware-aware modeling stage (Sec. III.4.1), the $\tanh(\cdot)$ non-linearity is implemented through a compact Taylor polynomial approximation to avoid LUT-based designs,

$$\tanh(a) \approx a - \frac{a^3}{3} = a - \frac{1}{3} \times a \times a \times a \tag{III.5}$$

which bounds the approximation error below the 8-bit quantization step (approximately 2^{-7}) in the operating range. The computation is scheduled over four PE cycles: two consecutive reads to form a, a third access to produce $a^3/3$, and a fourth to accumulate the final result into `REG_outPE`.

Weights, activations, and layer outputs use 8-bit fixed-point (1.7) to match the quantized network. To prevent overflow:

- CONV1 accumulators are 14 bits wide (sum of 64 signed 8-bit terms).
- CONV2 MAC accumulators are 21 bits wide (sum of 24 products of signed 8-bit operands).

All intermediate values are rounded/saturated back to 8 bits at layer boundaries.

Depending on the operation (plain convolution, plus bias, with/without padding), the CORE requires between 1 and $\texttt{max\_ccs\_inputCore} = 184$ cycles to process one input sample. If the system had to accept every new PDM bit immediately, the clock frequency would satisfy

$$f_{\text{clk}} > \texttt{max\_ccs\_inputCore} \times \text{ODR} = 184 \times 2.048\,\text{MHz} \approx 377\,\text{MHz}.$$

Instead, the `SIPO_IN` buffer decouples input acquisition from computation, so the minimum cycles available per input bit become

$$\texttt{min\_ccs\_inputSys} = \frac{\texttt{max\_ccs\_inputCore}}{\texttt{SIPO\_IN\_size}},$$

greatly relaxing the clock constraint.

From post-synthesis results in 130 nm CMOS, the dynamic power is $128.7\,\mu\text{W/MHz}$. To keep the on-sensor power below 1 mW (well under a typical MEMS microphone budget), we cap the operating frequency to about 7.6 MHz. With $\text{ODR} = 2.048$ MHz, this yields roughly three clock cycles per input bit on average, which—together with input buffering—meets real-time operation while staying within the power envelope.

The architecture reuses a single PE and a few byte-sized buffers under FSM control to implement the entire 1D-CNN with minimal hardware. CONV1 eliminates multipliers altogether by exploiting the binary nature of PDM; CONV2 uses a short MAC loop; the activation is computed via a low-cost cubic polynomial. Carefully sized accumulators guarantee correctness without resorting to wide data paths, and input buffering allows operation at a single-digit MHz clock, keeping dynamic power in the sub-milliwatt range appropriate for in-sensor integration.

Table III.2: *Artix–7 (XC7A35T) implementation at 6.5 MHz. Both designs process PDM at 2.048 MHz (OSR = 128) and produce 16 kHz PCM.*

	CIC-based Filter	**Proposed System**
Input Freq. [MHz]	2.048	2.048
Output Freq. [kHz]	16	16
OSR	128	128
Clk Freq. [MHz]	6.5	6.5
LUTs	744	917
FFs	812	361
DSPs	1	0
Dyn. Power [mW]	7.000	0.182

III.5.2 FPGA Results and Discussion of the Proposed PDM-to-PCM Converter

The neural PDM-to-PCM converter hardware was prototyped on a Xilinx Artix–7 (XC7A35T–CPG236–1) using the VIVADO design suite to validate the functional behavior and HW performance of the proposed design in real hardware. For a fair comparison, we also implemented the conventional multistage baseline of Fig. III.6 on the same device (CIC decimator followed by compensation and a low-pass FIR). Both designs were driven by the same testbench, with a PDM input rate of 2.048 MHz (OSR = 128) and a target PCM output of 16 kHz. The system clock was set to 6.5 MHz, which is the minimum frequency that guarantees real-time operation given the chosen input buffering (`SIPO_IN` = 62 bits).

Table III.2 reports post-implementation resource counts and dynamic power from VIVADO. The proposed design maps to slightly more LUTs than the CIC-based chain (917 vs. 744, $+23\%$), but it uses *far fewer* flip-flops (361 vs. 812, $\approx -56\%$) and no DSP blocks (0 vs. 1). Avoiding DSPs keeps the design portable across low-end FPGAs and aligns with the multiplier-free CONV1 and short, 8-bit MAC loop in CONV2. Most notably, the measured dynamic power is about one order of magnitude lower for the proposed design (0.182 mW vs. 7 mW at 6.5 MHz), corresponding to roughly $28\,\mu\mathrm{W/MHz}$.

Operating at 6.5 MHz with a 2.048 MHz input stream provides more than three clock cycles per incoming PDM bit on average, which—together with the `SIPO_IN` buffer—meets the worst-case cycle budget of the iterative CORE and ensures real-time decoding without stalls.

When driven with the same PDM sequences, the FPGA prototype produces PCM outputs that match the software model within the expected 8-bit quantization tolerance. Measured on a 1 kHz sinusoidal input, the converter achieves an SNR of $\approx$48 dB, consistent with the results reported in Sec. III. In the end-to-end evaluation with the KWS back end, the FPGA outputs preserve the 89% classification accuracy observed in software.

The resource profile highlights the key advantage of the neural interface for in-sensor computing: by consolidating decimation and anti-alias filtering into a learn-

Table III.3: *Synthesis results at sensor ODR = 2.048 MHz (TSMC 130 nm CMOS). Power from Cadence* JOULES *with SAIF; area and timing from post-synthesis reports.*

	CIC-based Filter	**Proposed System**
Clk Freq. [MHz]	123	6.5
Dyn. Power [μW]	2600	837
Area [mm^2]	0.080	0.086

able, shallow 1D-CNN, we remove the coefficient-heavy FIR chain and the single DSP usage of the baseline, cutting dynamic power by an order of magnitude while maintaining audio fidelity and KWS accuracy. The absence of DSPs and the modest register count make the design portable to a wide range of low-cost FPGAs and amenable to on-sensor ASIC integration in the next section.

III.5.3 ASIC Results and Discussion of the Proposed PDM-to-PCM Converter

To assess tight sensor–level integration, the converter was synthesized in a commercial 130 nm CMOS technology (TSMC 0.13 μm) using the Cadence digital flow. Power was estimated with Cadence JOULES from SAIF activity generated on the RTL testbench, while area and timing refer to post-synthesis reports. We set the MEMS microphone Output Data Rate (ODR) to 2.048 MHz (OSR = 128), which maps conveniently to a 16 kHz PCM rate and allows power-of-two decimation. All figures below refer to this ODR.

Table III.3 contrasts the proposed neural converter with a conventional multistage baseline (5th-order CIC with $64\times$ decimation, moving-average equalizer, and a 63rd-order FIR with an additional $2\times$ decimation). Thanks to the iterative microarchitecture and the absence of multipliers in CONV1, the proposed design meets real-time operation at a clock about $\sim$19$\times$ lower than the CIC-based chain (6.5 MHz vs. 123 MHz). This directly translates into a substantial dynamic-power reduction: 837 μW for the proposed system versus 2.6 mW for the CIC baseline (about $3.1\times$ lower at the target ODR). Area is comparable (0.086 mm^2 vs. 0.080 mm^2), with a modest $+7.5\%$ overhead that remains compatible with co-integration inside a digital MEMS microphone ASIC. Therefore, our proposal keeps total power and area well below 1 mW and 1 mm^2, compatible with ISC constraints.

From the synthesis reports, the proposed core closes timing at up to $\sim$200 MHz, establishing comfortable headroom for dynamic voltage and frequency scaling. Since the iterative engine requires roughly three core cycles per input bit on average (Sec.III. 4.1), the maximum interfaceable ODR is approximately $200/3 \approx 66$ MHz—far above what is needed for KWS, but useful for other sensing scenarios (Jamuna et al., 2014).

These results confirm that a learned, shallow 1D-CNN can replace the traditional CIC+FIR front end while staying within sub-milliwatt budgets and a tenth of a square millimeter in 130 nm CMOS. The significantly lower clock and power, together with comparable area, make the proposed PDM$\rightarrow$PCM interface an attractive building block for always-on, in-sensor TinyML pipelines.

These results confirm the feasibility of deploying the neural converter as part of an audio sensing front-end. Its low area and power profile, combined with high reconstruction fidelity and integration flexibility, make it a promising candidate for in-sensor machine learning applications, particularly in keyword spotting pipelines for wearable or battery-powered devices.

Chapter 4
Case 2: Low-Power In-Sensor Predictive Maintenance Based on Vibration Monitoring

IV.1 Publications Associated with This Chapter

The case study presented in this chapter builds upon peer–reviewed publications produced within the broader PhD research on Predictive Maintenance (PdM) and Anomaly Detection (AD) systems, conducted in collaboration with STMicroelectronics and University of Siena.

This research led to one Q1-journal papers and three conference proceedings. Readers may refer to the following for further details.

IV.1.1 Journal Articles

- Paola Vitolo et al. (2022a). "Low-Power Detection and Classification for In-Sensor Predictive Maintenance Based on Vibration Monitoring". In: *IEEE Sensors Journal* 22.7, pp. 6942–6951. DOI: 10.1109/JSEN.2022.3154479

IV.1.2 Conference Proceedings

- Paola Vitolo et al. (2021). "Low-Power Anomaly Detection and Classification System based on a Partially Binarized Autoencoder for In-Sensor Computing". In: *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pp. 1–5. DOI: 10.1109/ICECS53924.2021.9665479

- F. Spinelli et al. (2023). "Low-complexity Machine Learning Architecture for Hardware-aware True Random Number Generators Assessment and Continuous Monitoring". In: *2023 18th Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*, pp. 221–224. DOI: 10.1109/PRIME58259.2023.10161903

- Tommaso Addabbo et al. (2024). "Monitoring Hardware True Random Number Generators with Artificial Neural Networks: Problem Modeling and Training

Dataset Generation". In: *Applications in Electronics Pervading Industry, Environment and Society*. Ed. by Francesco Bellotti et al. Cham: Springer Nature Switzerland, pp. 291–296. ISBN: 978-3-031-48121-5

IV.2 Introduction

The rapid progress of AI—and Deep Learning (DL) in particular—is catalyzing the deployment of effective AD systems as a first step toward full PdM in Industry 4.0, where reliability gains and cost reductions are primary drivers (Ran et al., 2019). Although faults in industrial assets arise from heterogeneous electrical and mechanical causes, recent studies indicate that bearing issues alone account for roughly 30–40% of failures (Zhang et al., 2020). Early symptoms of incipient mechanical faults often manifest as weak, atypical vibration patterns that can be sensed by acoustic or inertial transducers (e.g., vibrometers, accelerometers) mounted on the machine body and relayed to a processing unit (Zhang et al., 2020). Because industrial environments are typically noisy and operating conditions vary, DL methods—by learning discriminative structure directly from sensor streams—tend to uncover subtle correlations between normal and anomalous behaviors more effectively than analytic or rule-based approaches, especially at an early stage when deviations are small (Ran et al., 2019; Yang et al., 2020).

Importantly, even when the monitored assets are not *portable*, practical PdM deployments often rely on *distributed* sensing points that must be retrofitted, replicated at scale across large facilities, or connected wirelessly as Industrial-IoT nodes. In such scenarios, the dominant bottlenecks are frequently node-level constraints—continuous (*always-on*) monitoring, communication bandwidth, installation and maintenance overhead (e.g., battery replacement), and local thermal/power/area budgets—rather than the availability of mains power at the plant level. Consequently, performing early inference near (or within) the sensor to reduce raw-data streaming and enable event-driven operation becomes a common design objective in edge-AI condition monitoring, including TinyML-based vibration anomaly detection on resource-constrained devices and low-power wireless sensing nodes for fault diagnosis in harsh environments (Arciniegas et al., 2025; Kolok et al., 2025; L. Wang et al., 2025; Losada et al., 2024). Moreover, the same architectural rationale extends beyond factory machinery to other distributed and mobile domains—such as rail/rolling stock and automotive onboard monitoring—where wiring is costly and per-node communication/power budgets remain constrained (Dziadak et al., 2022; Tang et al., 2023).

Modern PdM literature explores several DL families—CNNs, Generative Adversarial Networks (GANs), and Recurrent Neural Networks (RNNs)—each with distinct strengths and weaknesses (Ran et al., 2019). However, running such models continuously in real deployments is challenging: computational cost, memory footprint, and energy consumption are often incompatible with always-on monitoring budgets at the edge (Yang et al., 2020; Saufi et al., 2019). In this context, autoencoders (AEs) are particularly appealing: they can be trained *unsupervised* to reconstruct their inputs and thus flag anomalies via reconstruction error when the incoming data deviates from the learned notion of normality—an advantage when labeled fault data are scarce (Capra et al., 2020). The limitation, however, is intrinsic: AEs detect that an anomaly occurred but typically do not classify its type (Zhai et al., 2018). Related work has

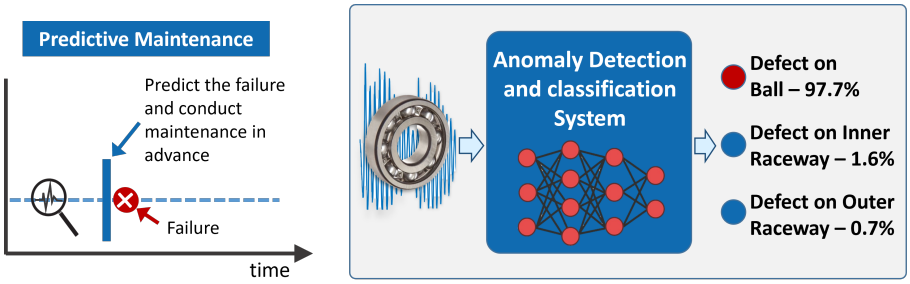

Figure IV.1: *Predictive-maintenance concept and system-level block diagram for bearing monitoring.* ***Left:*** *the machine is continuously monitored so that intervention occurs before the fault occurs.* ***Right:*** *the proposed pipeline processes the vibration signal with an anomaly detection and classification module; when an anomaly is flagged, an on-demand classifier is triggered to identify the fault type (e.g., defect on ball, inner raceway, or outer raceway) and report confidences.*

attempted to overcome this gap either with compact conventional Machine Learning (ML) classifiers or with deeper DL stacks, yet practical systems still struggle to balance implementation resources, throughput, and detection/classification accuracy (Ribeiro et al., 2020; STMicroelectronics, 2015).

In this chapter's case study, as shown in Fig.IV.1, we present a low-power Anomaly Detection and Classification (ADC) system for PdM that addresses these limitations with an in-sensor, edge-AI design. The main features are:

1. **Continuous vibration monitoring** of mechanical equipment with real-time AD (each input processed before the next arrives) at high ODR.
2. **Very high AD accuracy** and **on-demand anomaly classification** across nine fault/operating classes: a deep *convolutional AE* performs lightweight, always-on detection; upon anomaly, an *Anomaly Classifier* (AC) wakes up and classifies the event asynchronously.
3. **In-sensor computing** to reduce energy and bandwidth: a large fraction of the computation is placed *at or within the sensor* to curtail continuous data movement on the communication channel (Pan et al., 2022; STMicroelectronics, 2020).

Architecturally, as illustrated in Fig.IV.2, the proposed system is hybrid: a deeply quantized, low-area AE operates continuously near the sensor and also serves as a feature extractor for the classifier by exposing its encoder embedding. The AC—implemented on a low-power microcontroller (MCU)—is kept in deep sleep and activated on demand by the AE only when an anomaly is detected, thus gating power and compute in normal operation. The AE is expressly optimized for tight integration with the CMOS circuitry commonly co-packaged with inertial MEMS (A/D, filters, resamplers, etc.), enabling practical in-sensor AI (STMicroelectronics, 2020). Its design choices are summarized below:

- **Multilayer convolutional encoder/decoder** to achieve high detection accuracy with shallow depth relative to general-purpose DL.

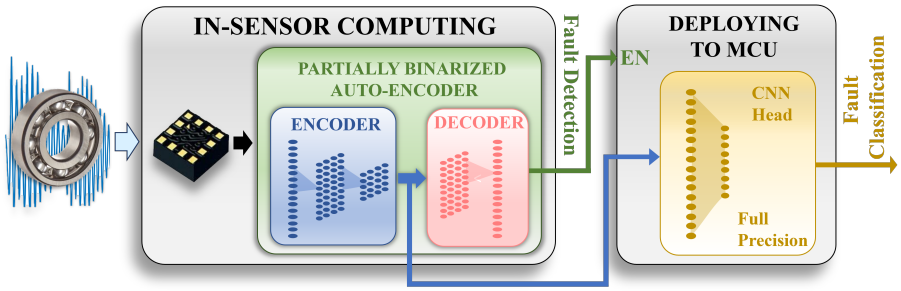

Figure IV.2: *Proposed hybrid in-sensor/MCU architecture for vibration-based predictive maintenance. A MEMS inertial sensor acquires the bearing vibrations and feeds a* partially binarized *convolutional auto-encoder that runs* ***in-sensor*** *to reconstruct the signal and perform anomaly detection at the sensor ODR. When the reconstruction error exceeds a threshold (*fault detected*), the encoder features are forwarded and a wake-up is issued to the MCU, which—normally in deep sleep—executes a* ***full-precision*** *CNN head to classify the fault. This split minimizes energy and bandwidth on the sensor while preserving classification accuracy. Extracted from (Vitolo et al., 2022a)*

- **Custom partial binarization** of selected layers: weights are binarized where accuracy is preserved, while activations remain in low-precision fixed point for the most sensitive stages—balancing accuracy and hardware cost.

- **Hardware-centric optimization** to limit resources via tailored datapaths and memory organization, rather than by aggressively pruning network depth (Antonio De Vita et al., 2020).

- **Classifier sharing**: thanks to the small encoder embedding, a single AC can be time-multiplexed across multiple AEs distributed on the machine, if desired.

As a representative application, we target bearing health monitoring and adopt the Case Western Reserve University (CWRU) dataset for training and validation (*Case Western Reserve University (CWRU) Bearing Data Center* 2020). The proposed system achieves state-of-the-art figures: 99.61% detection accuracy and up to 94.83% classification accuracy across 9 classes. On a Xilinx Artix-7 FPGA prototype, the AE consumes 122 mW total (15 mW dynamic) at 45 MHz, i.e., $\approx 333\,\mu\mathrm{W/MHz}$ dynamic, and supports MEMS ODRs up to 365 kHz, a key requirement for real-time monitoring of high-speed tools (e.g., drills, cutters). A 65 nm LP-HVT CMOS synthesis reports 138.62 μW/MHz dynamic, 0.49 mm^2 core area, and 230 MHz maximum frequency; analyses in 90 nm and 130 nm HVT nodes further support the feasibility of embedding the accelerator alongside sensor readout blocks for in-sensor computing. To the best of our knowledge, these results exceed the state of the art for systems with comparable constraints.

Section IV.2 reviews recent PdM/AD literature and positions our design. Section IV.3 details the AE/AC models. Section IV.4 presents the hardware architecture and implementation. Section IV.5 reports FPGA/ASIC results and comparisons.

IV.3 State of the Art

To endow AEs with *recognition* capabilities, recent contributions often augment AEs with conventional ML, which is computationally lighter than DL. A stream of works focuses on improving AE-based models. For instance, a Feature Distance–Stacked AE (FD–SAE) has been paired with a Support Vector Machine (SVM) to classify three anomaly types, but only when their effects on bearings deviate markedly from normal operation (Cui et al., 2021).

A deep generative approach based on a Variational AE (VAE) performs classification with a reduced set of labeled samples (Zhang et al., 2021). Wavelet-packet denoising combined with random forests attains 88.23% fault-diagnosis accuracy for rolling bearings under noise (Z. Wang et al., 2017). In parallel, 2D CNNs on vibration data further boost fault-classification accuracy (Magar et al., 2021).

Despite these advances, many models demand substantial compute and memory, making them ill-suited for resource-constrained embedded deployments. Cloud offloading is generally impractical for mission-critical assets because of latency, bandwidth, and reliability concerns; likewise, CPU/GPU platforms have silicon area, cost, and power profiles incompatible with on-machine integration (Capra et al., 2020). Specialized accelerators such as the Xilinx Deep Learning Processor Unit (DPU) support common DL operators but target high performance and remain too resource- and energy-hungry for extreme deep-edge scenarios (Lei et al., 2020).

Several FPGA designs (Tsukada et al., 2020; Maria et al., 2016; Coutinho et al., 2019) also instantiate large neuron counts, with commensurate resource and power budgets. Lightweight CNN families (e.g., Tiny-YOLO, MobileNet, ShuffleNet) reduce complexity (Fang et al., 2020; Yu and Lv, 2021; H. Liu et al., 2019; C. C. Chen et al., 2021), yet they still fall short of microwatt-level power envelopes required for always-on sensing.

A configurable neural architecture has been proposed to lower average power by operating a small, low-precision AE in the nominal case and scaling complexity/precision once an anomaly is detected (S. K. Bose et al., 2020). While attractive for detection, such gating is not ideal for PdM, which demands high sensitivity to weak, early-stage faults (Zhang et al., 2020; Yang et al., 2020).

In contrast, our approach emphasizes custom neural hardware to keep most computation *in sensor*, thereby minimizing system energy and communication bandwidth (Z. Zou et al., 2019) while supporting high-ODR inertial MEMS sensors (STMicroelectronics, 2015; STMicroelectronics, 2020).

IV.4 The Proposed Models for the Anomaly Detection and Classification System

The proposed ADC system is illustrated in Fig. IV.3.
It comprises two neural components:

- **Autoencoder (AE)** — always-on *anomaly detection*: composed of an *Encoder* followed by a *Decoder*; faults are flagged via reconstruction error.
- **CNN-based Classifier** — *fault identification*: when triggered, it processes the Encoder's embedding to classify the anomaly type.

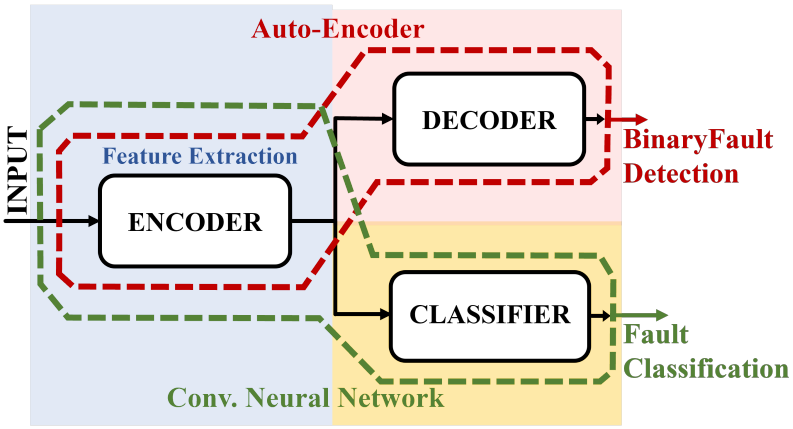

Figure IV.3: *Y-shaped neural architecture for predictive maintenance. A shared convolutional* ***Encoder*** *performs feature extraction and is reused by two heads: (i) with the* ***Decoder****, it forms an autoencoder for* binary fault detection *via reconstruction error; (ii) with the* ***Classifier****, it forms a CNN branch for* fault classification*. Sharing the encoder reduces memory/compute and allows the classifier to remain idle until an anomaly is detected. Extracted from (Vitolo et al., 2022a)*

The *Encoder* is shared by both models in a Y-shaped topology: It serves as a shared feature extractor for both the Decoder and the CNN. It runs continuously to produce features for AD; upon an anomaly, the same features are forwarded to the classifier head. This sharing eliminates duplicated computation and memory and enables power/compute gating of the classifier during nominal operation.

Both models adopt a *partial quantization* strategy previously validated by the authors on a resource-constrained Human Activity Recognition (HAR) pipeline (A. De Vita et al., 2020b; A. De Vita et al., 2020a). Although the present application, network sizes/topologies, and sensing modality differ, the same principle proved effective here for PdM. In particular, *all AE weights are binarized* to $\{-1, +1\}$ and stored with 1 bit, yielding an approximate 32:1 reduction in parameter memory compared with 32-bit floating-point, and replacing multiplications with sign-controlled ADD/SUB operations—substantially reducing area and dynamic power. In addition, the *most compute- or memory-intensive activations* are binarized, while sensitive stages retain low-precision fixed-point to preserve accuracy; *Batch Normalization* layers are inserted to stabilize training and mitigate quantization-induced drift (Sari et al., 2019). This combination delivers an accuracy–efficiency trade-off suitable for in-sensor deployment.

IV.4.1 Anomaly Detection Model

The proposed AE is shown in Fig. IV.4, and is organized as two sequential stages: an *Encoder* (Fig. IV.4a) and a *Decoder* (Fig. IV.4b). A convolutional design is adopted for both stages, which brings several advantages for the HW implementation of the AE:

- *Fewer parameters and lower resource usage* than an equivalent fully connected

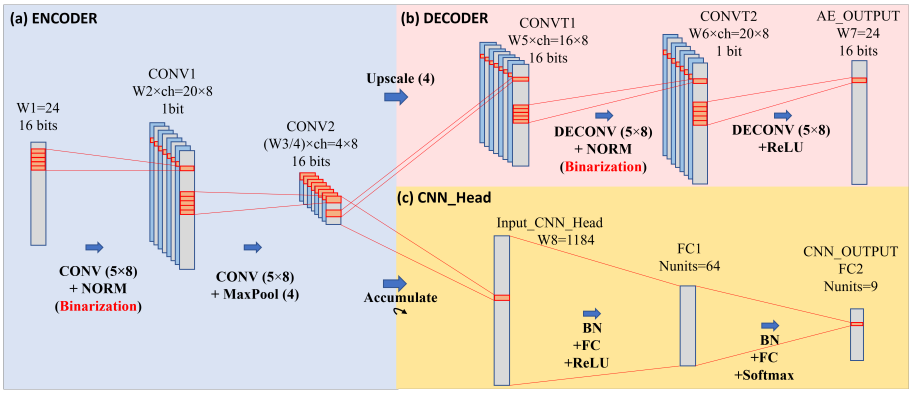

Figure IV.4: *Architecture of the proposed ADC model. (a)* ***Encoder****: two 1D convolutional blocks—*Conv(5×8)+BN *with binarization, followed by* Conv(5×8)+MaxPool(4)*—produce the shared features. (b)* ***Decoder****:* Upscale(4) *and two transpose-convolution blocks—*Deconv(5×8)+BN *with binarization and* Deconv(5×8)+ReLU*—reconstruct the 24-sample output. (c)* ***CNN_Head****: the encoder features are accumulated into a vector (*W_8=1184*) and passed through* BN+FC+ReLU *and* BN+FC+Softmax *to produce 9-class fault predictions. The encoder is shared by both branches, yielding a Y-shaped AE+classifier topology. Extracted from (Vitolo et al., 2022a)*

topology, substantially reducing memory and logic.

- *Simpler operators and shorter data paths* in the convolutional layers, which favor iterative HW realizations of CONV blocks with acceptable latency—even for sensors operating at very high ODR.

- *Smaller area and lower power* than a conventional dense network thanks to weight sharing and localized computation.

The encoder comprises two convolutional (CONV) layers followed by a Max-Pooling (MP) layer, acting as hierarchical feature extractors and a decimation filter, respectively. Each CONV uses 8 channels, kernel size 5, stride 1, and *no* zero padding. The input window length is W_1=24 samples. Inputs to the first layer are *not* binarized; they are represented with 16 bits, reflecting the output format of several commercial PdM sensors (STMicroelectronics, 2015).

Because padding is zero, the number of outputs per channel after CONV1 is $W_2 = W_1 - (5-1) = 20$. Thus, the CONV1 activation tensor has size $W_2 \times \mathrm{ch} = 20 \times 8 = 160$ samples. CONV1 is followed by a Batch Normalization (BN) layer and a binarization step, so each output can be represented with a single bit.

The binarization activation is the sign function,

$$y = \mathrm{sgn}(x) = \begin{cases} -1, & x < 0, \\ +1, & x \geq 0, \end{cases} \tag{IV.1}$$

Sign activation function which is used here because most of the subsequent computations are performed in CONV2.

Table IV.1: *Memory and operations required by the ADC model. Extracted from (Vitolo et al., 2022a)*

	Layer	Parameters [bytes]	Op. per window
ENCODER (24 inputs)	*Conv1*	21	960 ADDs
	Norm1	16	160 SUBs
	Conv2	56	5,248 ADDs
	Max Pool	0	128 SUBs + COMP
DECODER (24 inputs)	*Upscale*	0	0
	ConvT1	56	6,560 ADDs
	Norm2	16	160 SUBs
	ConvT2	21	984 ADDs
CNN Head (600 inputs)	*BN*	9,472	4,736 ADDs + 4,736 MULTs
	FC1	303,360	75,841 ADDs + 75,776 MULTs
	BN	512	256 ADDs + 256 MULTs
	FC2	2,340	585 ADDs + 576 MULTs

Layer CONV2 applies a bank of 8 filters of size 8×5; its output activations have size $W_3 \times \mathrm{ch} = 16 \times 8 = 128$ samples. The following Max-Pooling layer has pooling size 4, therefore its output is $W_3/4 \times \mathrm{ch} = 4 \times 8 = 32$ samples.

The decoder mirrors the encoder. It first applies an *UpSampling* layer that quadruples the temporal length of its input, and then a transpose-convolution (ConvT) that implements the inverse filtering. The input size of the first transpose-convolution is $W_5 \times \mathrm{ch} = 16 \times 8$; uniform padding yields an intermediate size of 24×8, after which a convolution with kernel size 5 and 8 channels is applied. The resulting activation size is $W_6 \times \mathrm{ch} = 20 \times 8 = 160$ samples. As in the encoder, ConvT1 is followed by BN and binarization, so each output can be encoded with 1 bit. ConvT2 uses the same padding and convolution settings as ConvT1, but employs the ReLU activation,

$$y = \mathrm{ReLU}(x) = \begin{cases} 0, & x < 0, \\ x, & x \geq 0, \end{cases} \tag{IV.2}$$

ReLU activation function for the final reconstruction.

Table IV.1 reports the AE complexity, detailing for each layer the parameter memory and the number of operations per window. To minimize HW resources in the complete system, the *Encoder* also serves as the shared feature extractor for a shallow classifier; together they form the CNN head for fault classification, sharing a large portion of resources with the AE.

IV.4.2 Fault Classification Model

Figure IV.4c depicts the **CNN_Head**, which—together with the shared encoder of Fig. IV.4a—forms the classifier branch. In this branch the encoder provides the feature extractor (two partially binarized CONV layers, one normalization layer, and one MaxPool), while the head itself consists of two *full-precision* fully connected

(FC) layers followed by a Softmax. The first FC layer (**FC1**) has 64 neurons with the ReLU activation of (IV.2); the second (**FC2**) has 9 neurons, matching the number of fault/operating classes considered.

Unlike the AE, a 24-sample input window does not provide enough temporal context to reliably disambiguate fault types. With $W{=}24$, the mean classification accuracy is about $55\,\%$. Increasing the input span improves performance markedly: using $W{=}336$ samples yields $91.92\,\%$ accuracy, and performance peaks at $94.83\,\%$ with $W{=}600$ samples. In the final design we therefore adopt $W{=}600$ for the classifier.

The CNN_Head is deployed on an STM32L476RG microcontroller using the X-CUBE-AI (v6.0.0) toolchain, which converts the pre-trained network to C, compiles it, and profiles it on the target. A single inference requires $79\,120$ MACC operations; with an average of 8.69 clock cycles per MACC at $80\,\mathrm{MHz}$, the end-to-end latency is $8.59\,\mathrm{ms}$—incurred only after the AE raises an anomaly. The STM32 consumes approximately $120\,\mu\mathrm{A\,MHz^{-1}}$ in run mode. The layer-wise parameter memory and arithmetic counts for the head are reported in Table IV.1. Because the head uses single-precision floating point and a comparatively large memory footprint, implementing it *in-sensor* is not practical; instead it is naturally realized as software on a low-cost MCU (or as a companion block on an FPGA) operating in tandem with the AE.

To reconcile the encoder's native window ($W{=}24$) with the classifier's requirement ($W{=}600$) while keeping the ADC compact and encoder-shared, we accumulate the 24-element MaxPool output of the encoder in the MCU RAM across 25 successive windows (on each AE trigger), thus forming a $24 \times 25 = 600$-element input vector for the CNN_Head. This arrangement (i) enables easy in-field fine-tuning of the classifier without modifying the in-sensor AE (e.g., to compensate for wear or operating-point changes), and (ii) allows time-sharing of a single CNN_Head across multiple sensors that expose the same encoder features but are mounted at different locations on the monitored apparatus.

IV.4.3 Dataset Used and Model Results for the Proposed Anomaly Detection and Classification System

To validate the proposed ADC, we adopt the bearing dataset from Case Western Reserve University (CWRU, Cleveland, OH, USA) (*Case Western Reserve University (CWRU) Bearing Data Center* 2020). The test rig records motor-bearing vibration at 12 kHz and 48 kHz using accelerometers mounted on the motor housing with magnetic bases; electrical-discharge machining introduces defects of four diameters on the rolling elements, outer raceway, and inner raceway. Speeds are 1797, 1772, 1750, and 1730 rpm, with loads of 0, 746, 1492, and 2238 W.

For this work we extract two aligned datasets: **(i)** $20{,}328$ windows of length $W{=}24$ samples for the AE, and **(ii)** $2{,}637$ segments of length $W{=}600$ for the CNN_Head, all stored at 16 bits. In the classifier, 9 fault/operating classes are considered (ball, inner, and outer raceway, each with three defect sizes); the composition is summarized in Table IV.2.

The AE and the CNN_Head are implemented in TensorFlow; binarized layers are handled with Larq (Abadi et al., 2016; *LARQ Documentation* 2021). The AE dataset is split into **75%** training, **12.5%** validation, and **12.5%** test. Because the CWRU corpus contains both normal and faulty data, we follow the anomaly-detection proto-

Table IV.2: *Subset of the CWRU dataset used for the CNN. Extracted from (Vitolo et al., 2022a)*

Fault Location	Diameter [inch]	# Samples	Sample Length
Ball	0.007	293	600
	0.014	293	600
	0.021	293	600
Inner raceway	0.007	293	600
	0.014	293	600
	0.021	293	600
Outer raceway	0.007	293	600
	0.014	293	600
	0.021	293	600

col in (Yamanaka et al., 2019). The AE is trained for 50 epochs with batch size 256. We report accuracy and Area Under the ROC Curve (AUC) on the test set; AUC is threshold-independent and evaluates the detector irrespective of the operating point. Averaged over 10 trials, the AE attains **AUC = 0.99** and **accuracy = 99.61%**.

For the classifier, the CWRU subset is split into **80%** training, **10%** validation, and **10%** test. We use categorical cross-entropy and Adam, train for 200 epochs with batch size 128, and average results over 10 runs. The final **mean accuracy is 94.83%**. Empirically, classification accuracy improves with temporal context: it is $\sim$55% with $W{=}24$, rises to **91.92%** with $W{=}336$, and peaks at **94.83%** with $W{=}600$.

Table IV.3 contrasts our partially binarized CNN classifier with a recent 1D-CNN approach from the literature (C. C. Chen et al., 2021); for context we also refer to lightweight 2D CNN families (MobileNet, ShuffleNet V2) reported in (H. Liu et al., 2019). The 1D-CNN in (C. C. Chen et al., 2021) operates directly on 1D vibration, thereby avoiding STFT-based 2D time–frequency transforms (often required by 2D methods (H. Liu et al., 2019)) and yields strong accuracy. Nevertheless, our head requires *orders of magnitude* fewer multiply operations and parameter bytes than (C. C. Chen et al., 2021), thanks to partial binarization and the shared-encoder design. The observed $\sim$4.47% reduction in accuracy relative to (C. C. Chen et al., 2021) (from 99.3% to 94.83%) is an acceptable trade-off for the intended hardware deployment, where detection is performed in-sensor and classification is invoked on demand at the MCU.

IV.5 Hardware Architecture of the Proposed Anomaly Detection and Classification System

The HW design strategy for the AE accelerator targets the *minimum* number of circuit elements and exploits extensive resource sharing and iterative processing to meet the tight area–power budgets of in-sensor deployment, while still interfacing in real time with ODR sensors.

With *partial binarization*, a single bit encodes the weights and the binarized activations of selected layers; non-binarized quantities are represented in fixed point.

We adopt a **16-bit** fixed-point format (**Q8.8**) for those values, which strikes a good balance between code length and accuracy. These quantization choices shrink the memory footprint of convolution kernels and partial sums so they fit in internal registers or distributed on-chip memories (e.g., FPGA block/distributed RAM), avoiding slower and more energy-hungry access to higher-level or off-chip memory.

Weight binarization converts multiply–accumulate (MACC) operations into sign-controlled additions/subtractions. Thus each CONV/CONVT layer effectively computes

$$\sum_{i=1}^{N} w_i\, x_i + b \;=\; \pm x_1 \pm x_2 \pm \cdots \pm x_N \;+\; b, \tag{IV.3}$$

Add–subtract realization of a convolution with binarized weights (MACC-free form). where w_i and x_i are the weight and input to a neuron, respectively, and b is the bias.

Because batch normalization (BN) is immediately followed by binarization, the BN layer can be implemented without a full-precision multiplier. For a generic input x, BN produces

$$y \;=\; \alpha x + \beta \;=\; \alpha\left(x + \frac{\beta}{\alpha}\right).$$

Since the subsequent activation is the sign function (constraining y to ± 1), only the *sign* of α and the value $\gamma = \beta/\alpha$ are needed; these are encoded with 1 bit and 16 bits, respectively, eliminating the multiply while preserving the binarized output.

IV.5.1 Hardware Description of the Proposed Modules

Figure IV.5 schematizes the HW architecture of the AE. It consists of two modules that implement, respectively, the *encoder* and the *decoder*, a *FIFO* used to buffer data exchanged between the two modules, and a *Control Unit* (CU) that orchestrates all operations. The encoder and decoder operate *asynchronously* with respect to each other so that the encoder can accept a new input window without waiting for the decoder to complete, thereby increasing the maximum input data rate (IDR).

Figures IV.5(b) and IV.5(c) show the implementation diagrams of the encoder and decoder modules. Each module is composed of a local control unit (ENC_CU or DEC_CU) and a computational core (ENC_CORE or DEC_CORE) that iteratively ex-

Table IV.3: *Model comparisons. Extracted from (Vitolo et al., 2022a)*

	Proposed CNN	**C. C. Chen et al. (2021)**
Op. per window		
ADDs	243,530	2,537,306
MULTs	81,344	2,366,144
COMP	3,200	85,232
Memory [bytes]	408,221	1,058,568
Average Accuracy [%]	94.83	99.3

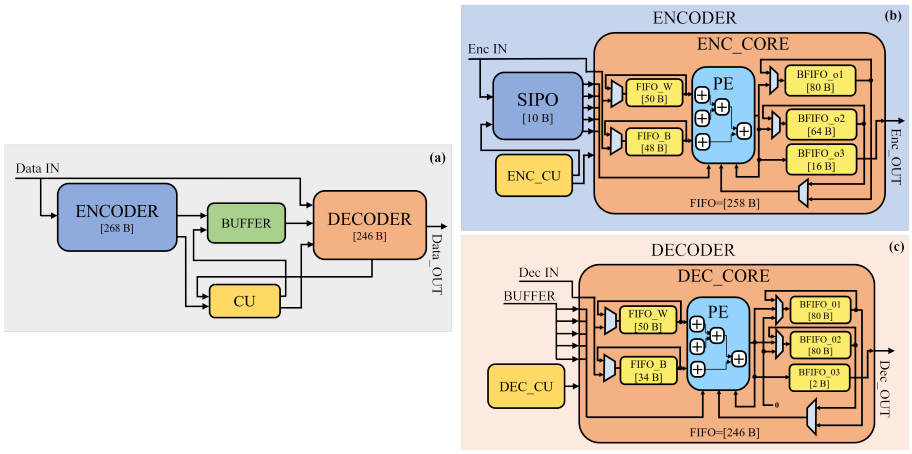

Figure IV.5: *Hardware architecture of the autoencoder accelerator.* ***(a)*** *Top-level organization: the* ENCODER *and* DECODER *exchange data through a FIFO* BUFFER *under a global Control Unit (*CU*); the two modules run asynchronously so the encoder can accept new inputs while the decoder is still processing. Byte counts indicate embedded FIFO/memory sizes (ENCODER: 268 B; DECODER: 246 B).* ***(b)*** *Encoder module: local controller (*ENC_CU*),* SIPO *for the receptive field, and* ENC_CORE *with FIFOs for weights (FIFO_W, 50 B) and biases (FIFO_B, 48 B), a Processing Element (PE), and three output circular buffers (BFIFO_o1/o2/o3: 80/64/16 B).* ***(c)*** *Decoder module: local controller (*DEC_CU*) and* DEC_CORE *with FIFOs for weights (50 B) and biases (34 B), the PE, and output buffers (BFIFO_o1/o2/o3: 80/80/2 B). Extracted from (Vitolo et al., 2022a)*

ecutes the operations of the encoder/decoder. The encoder also includes a *SIPO* (serial–in/parallel–out) buffer to hold the receptive field of the first CONV layer. Both cores comprise: (i) a *processing element* (PE), (ii) multiplexers that select the proper PE inputs, (iii) FIFOs, and (iv) embedded memories to store all temporary variables needed for local computation. Circular buffers based on FIFOs are used to store inputs, partial outputs, and network parameters.

Taking into account the design choices:

- the kernel size and the number of channels of each CONV layer are 5 and 8, respectively;
- weights and biases are encoded with 1 bit and 16 bits, respectively;
- the BN parameters, α and γ, are encoded like weights and biases, respectively;
- the numbers of weights, biases, and BN parameters stored in the *encoder* are:

$$\text{CONV1}_w + \text{BN1}_\alpha + \text{CONV2}_w = 5 \times 8 + 5 \times 8 \times 8 = 400,$$

$$\text{CONV1}_b + \text{BN1}_\gamma + \text{CONV2}_b = 8 + 8 + 8 = 24;$$

- the numbers of weights, biases, and BN parameters stored in the *decoder* are:

$$\text{CONVT1}_w + \text{BN2}_\alpha + \text{CONVT2}_w = 5 \times 8 \times 8 + 5 \times 8 = 400,$$

$$\text{CONVT1}_b + \text{BNT1}_\gamma + \text{CONVT2}_b = 8 + 8 + 1 = 17.$$

Consequently, the encoder FIFOs allocate 400 bits for weights and 24×16 bits for biases; the decoder FIFOs allocate 400 bits for weights and 17×16 bits for biases.

Each CONV layer can start as soon as a number of input samples equal to the kernel size is available, whereas the MaxPool must wait for a number of inputs equal to its pooling size. We therefore adopt an *iterative* schedule that does not wait for an entire window but only for what is strictly necessary to begin processing. For example, CONV1 starts when 5 of the 24 input samples are ready. This reduces the SIPO from 24×16 bits to 5×16 bits (a **79%** reduction). The FIFOs for partial results have the following sizes: CONV1 40×16 bits, CONV2 32×16 bits, MaxPool 8×16 bits, CONVT1 40×16 bits, and CONVT2 40×16 bits, for an **overall reduction of 74%**. As a result, ENC_CORE and DEC_CORE embed **268 bytes** and **246 bytes** of FIFO memories, respectively.

FIFOs that store weights and biases are initialized via an *external* data stream during setup; afterwards, the CU configures them as circular buffers for the entire run. The memories used for partial results act as FIFOs when written and as circular buffers when read as inputs by the following stage.

Figure IV.6 depicts the PE, shared by encoder and decoder. It is a three-level adder tree implemented in 16-bit fixed point (Q8.8). As in (IV.3), the PE performs a dot product between two length-5 vectors (`DIN` and `W`, with binarized entries) and adds either a bias or the previous partial sum. Adders A1–A3 are 16-bit units with the *sign-controlled add/subtract* logic for binarized multiplications; A4 and A5 are 16-bit adders. The tree output can be iterated through the `RES` mux to accumulate the result and realize the convolution in (IV.3) for each layer. The activation functions of (IV.1) and (IV.2) are implemented by the `BIN` and `RELU` muxes, driven by the adder tree and selected by the `ACT` and `RES` signals when the dot product is complete. The PE also performs the comparisons required by the MaxPool layer via the `MP` mux. In the decoder, padding and upscaling are carried out by inserting 8 zeros or replicating the previous 8 samples into buffers `BFIFO_o1` and `BFIFO_o2` in Fig. IV.5(c).

IV.5.2 Dynamic Generation of the Layers

To minimize HW resources, all layer operations are executed by *reusing* the single PE available in each module. Two control units, **ENC_CU** and **DEC_CU**, manage PE inputs/outputs and the correct placement of partial results into FIFOs. Both CUs are implemented as Finite State Machines (FSMs) with one *state per layer*; each state contains *sub–states* that sequence the dot–product and post–processing steps.

For CONV1 in Fig. IV.5, the controller uses two sub–states, each executed in one clock cycle: (i) a *dot–product* cycle, where the five input samples in the `SIPO` and the five weights in `FIFO_W` are combined and the bias from `FIFO_B` is added; and (ii) a *BN + binarization* cycle, where the previous result is offset by the BN parameter γ (from `FIFO_B`) and then binarized using the sign of α (from `FIFO_W`); the result is written to `BFIFO_o1`. This computes the first CONV1 output for one channel. Repeating the two sub–states for all 8 channels yields, for 5 inputs, a total of

$$2 \times 8 = 16 \text{ clock cycles}$$

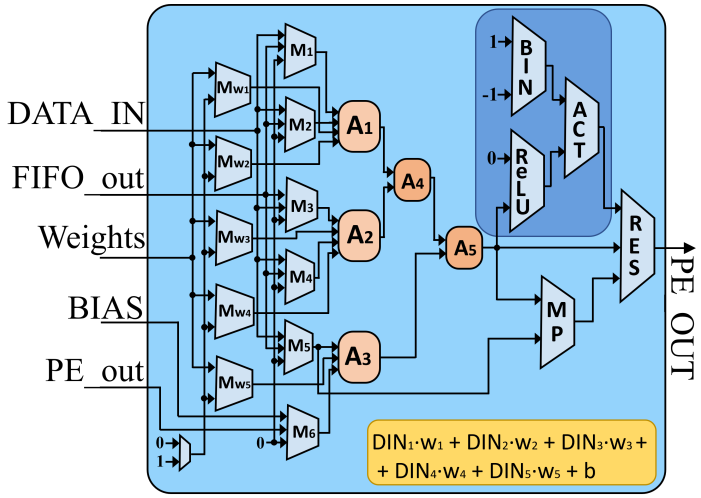

Figure IV.6: *Processing Element (PE) used in both encoder and decoder. Five input samples are combined with binarized weights via sign-controlled add/sub paths (multiplexers $M_{w1}\ldots M_{w5}$), then reduced by a three-level adder tree (A1–A5). The PE supports bias/partial-sum addition,* sign *binarization (*BIN*), ReLU activation, and result feedback (*RES*) to accumulate convolutions as in* (IV.3)*. A comparator path (*MP*) performs max-pool decisions. The block thus realizes the dot product* $\mathrm{DIN}_1 w_1 + \cdots + \mathrm{DIN}_5 w_5 + b$ *without explicit multipliers. Image extracted from (Vitolo et al., 2022a)*

to produce one full CONV1 output vector.

After CONV1, the remaining layers require the following cycles: **CONV2** 72, **MaxPool** 32, **CONVT1** 72, and **CONVT2** 9. Accounting for a single cycle to transition between FSM states, the **ENC_CORE** needs **123** cycles to process one input sample and, due to the final MaxPool of size 4, produces one output every 4 input samples.

The **DEC_CORE** is fed asynchronously by the encoder through the buffer and delivers an output in **9** to **83** cycles, depending on whether padding or upscaling is required.

Overall, the ADC can accept new input data every **123** cycles and processes the **16** input samples needed for fault detection in a minimum of **3632** clock cycles.

Since **ENC_CORE** and **DEC_CORE** operate in parallel, the maximum sensor ODR supported by the system is ultimately limited by the encoder processing time.

IV.6 Synthesis and Implementation Results

The proposed design has been prototyped on a Xilinx Artix–7 FPGA (device `XC7A35T-FGG484-1`) using the Vivado IDE. Figure IV.7 shows the test–board setup adopted to evaluate the system.

The platform consists of a compact Digilent *Cmod A7–35T* module, on which the **AE** accelerator is implemented, and an *STM32F401RE* microcontroller that hosts the **classifier** according to the proposed HW/SW hybrid scheme. The MCU also manages data transfer from the source to the FPGA through a custom, lightweight SPI–like

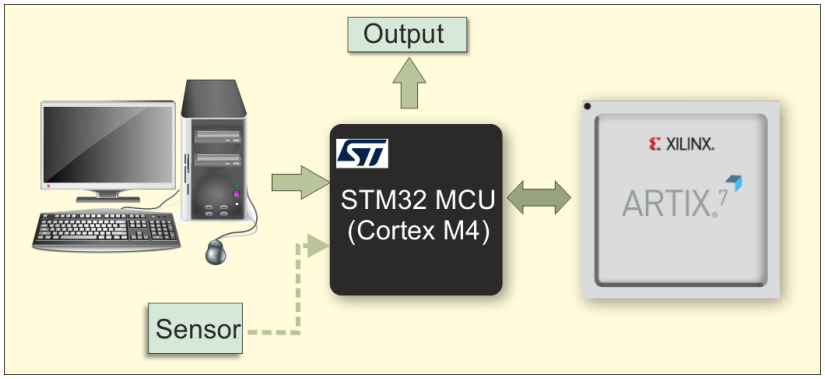

Figure IV.7: *Test-board setup for the HW/SW prototype. The STM32 (Cortex-M4) runs the CNN classifier and orchestrates data flow; inputs come either from a PC (replaying the dataset) or from a live sensor (dashed path). A lightweight SPI-like link connects the MCU to the Artix-7 FPGA, which hosts the autoencoder accelerator. Classification results are returned to the host as the system output. Extracted from (Vitolo et al., 2022a)*

interface.

As data source we use either (i) a PC, which stores and streams the test dataset, or (ii) a sensor front end such as the *X–NUCLEO–IKS01A1* shield equipped with the *LSM6DS0* IMU.

In addition to the FPGA prototype, we report ASIC synthesis results targeting TSMC $130\,\mathrm{nm}$, $90\,\mathrm{nm}$ and $65\,\mathrm{nm}$ CMOS standard–cell technologies using the Cadence toolchain, in order to assess the in-sensor integration cost relative to the other logic typically co-packaged with MEMS sensor circuitry.

IV.6.1 FPGA Implementation Results and Comparison

TablesIV.4 report the details of the FPGA implementations of our design and compare them against two recent related works (Tsukada et al., 2020; Belabed et al., 2020). Because the performance of these systems is ultimately governed by a three–way trade–off—(i) the ability to sustain high sensor output–data rates (ODR), (ii) power consumption, and (iii) mapped physical resources—we introduce a simple Figure of Merit (FoM) to enable direct comparisons. The *ODR–on–Power–and–Resources* (**OPR**) is defined as

$$\mathrm{OPR} = \frac{\mathrm{ODR}}{\mathrm{Tot.\,Power} \times \mathrm{Resources}}, \tag{IV.4}$$

where *ODR* is the real-time ODR supported by the implementation; *Tot. Power* is the total power at that ODR; and *Resources* denotes the physical resources used by the design—e.g., LUTs/FFs/BRAMs/DSPs for FPGA mappings or occupied area for ASICs.

Figure IV.8 plots the OPR of our design as a function of the ODR up to the maximum ODR supported by the prototype; the inset highlights the region around $\mathrm{ODR} = 1\,\mathrm{kHz}$ to facilitate a like-for-like comparison with (Tsukada et al., 2020;

Table IV.4: *FPGA Results and Comparison. Extracted from (Vitolo et al., 2022a)*

	Belabed et al. (2020)	**Tsukada et al. (2020)**	**Proposed AE**
HW Classifier	Yes	No	No
Accuracy [%]	–	–	**99.61**
AUC	–	0.952	**0.99**
HW platform	Zynq 7020	Zynq 7020	Artix 7
Freq. [MHz]	100	100	**45**
ODR @ Freq [kHz]	1.16	1	**365**
Dyn. Power [μW/MHz]	22360	–	**333**
Total Power @ ODR [W]	2.4	3.1	**0.122**
# of LUTs	38840	13725	**2449**
# of LUTRAMs	1916	–	**211**
# of FFs	43630	12342	**2319**
# of BRAMs	28	155	**0**
# of DSPs	46	72	**0**
OPR [Hz/W]	0.0059	0.0123	**1.96**

Belabed et al., 2020). Our AE achieves an AUC of 0.99, which is higher than the single accuracy metric reported by (Tsukada et al., 2020; Belabed et al., 2020). The curve in Fig. IV.8 shows the advantage of our approach: the implementation uses only

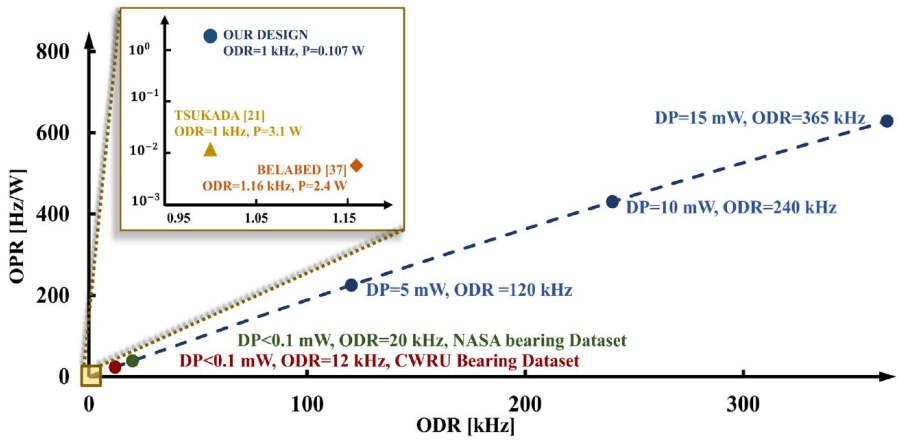

Figure IV.8: *OPR (Eq. IV.4) of the proposed AE versus sensor ODR. The inset zooms the $\sim$1 kHz region and compares against prior FPGA designs: Tsukada et al. (2020) (1 kHz, $P = 3.1$ W) and Belabed et al. (2020) (1.16 kHz, $P = 2.4$ W). At 1 kHz our design (total $P = 0.107$ W) achieves orders-of-magnitude higher OPR. Extracted from (Vitolo et al., 2022a)*

Table IV.5: *Comparison with Xilinx DPU (Lei et al., 2020). Extracted from (Vitolo et al., 2022a)*

	DPU core (B512)	Proposed AE
Power [W]	5.718	**0.122**
# of LUTs	36458	**2449**
# of FFs	41744	**2319**
# of BRAMs	77.5	**0**
# of DSPs	124	**0**

2449 LUTs and **2319 FFs** and deliberately avoids DSPs and BRAMs to keep the results as platform-independent as possible, yet it sustains a **maximum sensor ODR of 365 kHz**. Although our clock frequency (45 MHz) is lower than that of competing designs, the achievable ODR is more than two orders of magnitude higher than the counterparts. Such a high ODR enables demanding PdM scenarios that require real-time monitoring of high-speed mechanical parts (e.g., drills and cutters), and it is compatible with ultrasonic MEMS sensing (STMicroelectronics, 2020; Murphy, 2025) for applications such as pressure-leak detection, bearing condition monitoring, gear-mesh analysis, and pump-cavitation monitoring. Moreover, if the operating frequency is reduced to set the ODR to 1 kHz—i.e., matching the conditions of the alternatives in Table IV—the total power of our solution becomes **107 mW**, which remains significantly lower than the competing designs.

A different comparison can be drawn against FPGA Application Processing Units (APUs) such as the Xilinx *Deep Learning Processing Unit* (DPU) (Lei et al., 2020), which accelerates high-performance CNNs (e.g., GoogLeNet, ResNet, MobileNet) on Zynq SoCs. The DPU IP allows certain reductions—e.g., fewer DSP slices, less block RAM/UltraRAM, fewer cores, or smaller convolution engines—to meet spe-

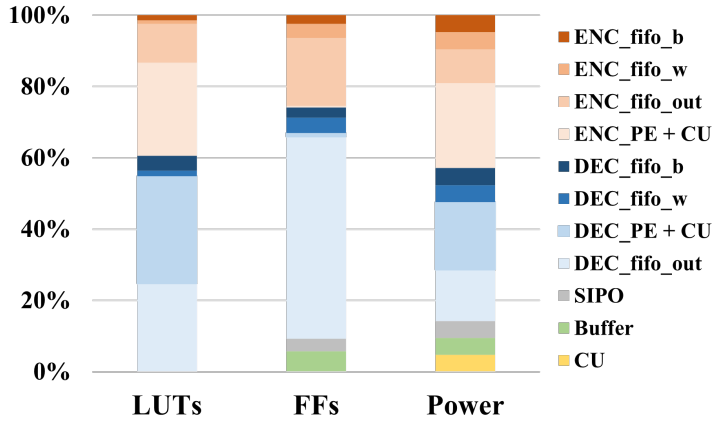

Figure IV.9: *FPGA breakdown of the proposed AE (Xilinx Artix-7, 45 MHz). Extracted from (Vitolo et al., 2022a)*

Table IV.6: *Standard-cells synthesis comparisons. Extracted from (Vitolo et al., 2022a)*

	Proposed AE				**S. K. Bose et al. (2020)]**
	130 nm HVT	90 nm LP HVT	65 nm GP	65 nm LP HVT	65 nm
Acc [%]	–	–	–	–	–
Max Freq. [MHz]	100	150	250	200	–
Max Sensor ODR [MHz]	0.81	1.21	2	1.63	–
Dyn. Power [μW/MHz]	599	210	120	138	–
Total Power @ ODR=20 kHz [μW]	1474	519	1512	341	744/ 12*
Area [mm^2]	3.54	1.30	0.51	0.49	0.57
OPR @ ODR=20 kHz [Hz/(μW·mm^2)]	3.83	29.64	1.55	1.20	47.16/ 2924*

*Active phase / inactive phase.

cific constraints. Nevertheless, even the smallest convolution configuration, **B512**, requires **36,458 LUTs**, **41,744 FFs**, **77.5 BRAMs**, and **124 DSPs**, with a power of **5.718 W** on an UltraScale+ ZCU102 (Table IV.5). In addition, one must provision the APU with program memory and handle interrupts, data transfers, and storage of input/temporary/output buffers, resulting in an overall resource demand far greater than in our project.

Finally, Fig. IV.9 breaks down the FPGA implementation of the AE in terms of utilization and power. Approximately **56%** of the mapped LUTs and **43%** of the power are consumed by the Processing Elements (PEs) and Control Units (CUs) of the encoder and decoder. About **89%** of the FFs are occupied by the FIFOs, whose count is the true limiting factor of the proposed architecture.

IV.6.2 ASIC Synthesis Results and Comparison

Table IV.6 summarizes standard–cell synthesis results for 130 nm (the node commonly used for commercial MEMS-sensor ASICs), 90 nm, and 65 nm CMOS technologies. For the 130 nm and 90 nm cases, as well as for one 65 nm flavor, we employed High-Voltage-Threshold (HVT) cell libraries because their much lower leakage is critical at low operating frequencies (Antonio De Vita et al., 2020). For ease

of comparison, a 65 nm General-Purpose (GP) library has also been included. Using Cadence Joules with SAIF activity, HVT libraries reduce leakage power by about **77%** relative to GP, at the expense of a *maximum* frequency decrease of roughly **10%**. The resulting maximum operating frequencies (100–250 MHz) correspond to sensor ODRs in the MHz range, which— although higher than required in our current application—provide an indication of the scalability margin of the proposed accelerator with future sensor technology.

Table IV.6 also reports the *total* power when the ODR is set to 20 kHz, which maps to an AE operating frequency of 2.46 MHz. Note the significant contribution of static power at 65 nm when using the GP library, due to leakage, compared with the HVT counterpart.

A direct comparison with prior art is non-trivial: only a few works target low-power, integrated AE-based PdM, and many papers omit full implementation figures. From the literature, we consider ADEPOS (S. K. Bose et al., 2020; Kar et al., 2020), one of the few designs implemented in 65 nm CMOS. Although not strictly an apples-to-apples comparison, ADEPOS reduces power from a *maximum* of 744 µW to a 12 µW standby level by *fully activating only after an anomaly is detected*, dynamically increasing complexity and accuracy for about 1% of the device lifetime (based on the dataset in (Xilinx, 2018; *NASA Prognostics Data Repository* 2025) and an ODR of 20 kHz). Under those conditions, ADEPOS achieves $\mathrm{OPR} = 47.16\ \mathrm{Hz\,W^{-1}\,mm^{-2}}$ (and 2924 in the inactive phase).

By contrast, our design operates at a *constant* high-accuracy level—more appropriate for PdM where early, weak anomalies must be detected—with a total power of **341 µW** at 20 kHz ODR and an occupied area of **0.49 mm²**, yielding $\mathrm{OPR} = 120\ \mathrm{Hz\,W^{-1}\,mm^{-2}}$. To the best of our knowledge, these results advanced the state of the art for this class of systems.

These results demonstrate that pushing feature extraction and anomaly detection into the sensor is a practical and energy-efficient path to robust PdM at the deep edge.

Future work will explore (i) multi-sensor fusion (e.g., inertial, acoustic/ultrasonic) to improve sensitivity under varying operating regimes; (ii) robustness to domain shift via self-/continual learning and adaptive thresholding in the field; (iii) further compression (selective binarization/pruning) and architectural co-design for newer technology nodes; and (iv) extending the pipeline to fault-severity estimation and remaining useful life (RUL) prediction while preserving the in-sensor energy envelope.

IV.6.3 Discussion on Architectural Partitioning

The proposed system adopts a hybrid partitioning, with an always-on AE operating close to the sensing interface and a higher-level classifier (CNN_Head) activated only on demand. This choice is motivated by the results above, which show that early anomaly detection reduces continuous data streaming and enables event-driven activation of downstream processing.

A natural alternative would be to run the AE on the MCU. While feasible in principle, this option would typically require keeping the MCU active continuously to sustain always-on monitoring, reducing the effectiveness of deep-sleep operation and increasing average energy consumption at the sensing node. In distributed Industrial-IoT deployments, where many sensing points must be installed and maintained, aver-

age node power and duty-cycling capability remain key scalability drivers.

Another alternative would be to map both the AE and the CNN_Head onto an FPGA. This could further reduce external communication, but it generally trades increased static power and cost for implementation flexibility and prototyping convenience. Moreover, executing the full pipeline continuously at the sensor node is not always necessary: keeping only the compact AE always active and triggering the classifier sporadically better matches the event-driven nature of predictive-maintenance monitoring and the constraints-first objectives of this thesis. Additionally, in this work, the FPGA option is primarily relevant as a prototyping vehicle rather than as a target for sensor-integrable deployments.

Finally, a full custom-ASIC implementation of both the AE and the CNN_Head is also possible in principle. However, executing the entire pipeline always-on at the sensor node is not always necessary: the classifier is only required sporadically, once an anomaly has been detected. Integrating the CNN_Head together with the always-on AE would increase silicon area and power budgets to support a function that is intrinsically event-driven and infrequently invoked. In this sense, the marginal benefit of moving the full classifier on-chip would not justify the additional hardware cost for the targeted monitoring regime.

Moreover, keeping the CNN_Head outside a fixed-function ASIC preserves system flexibility: in real deployments, the set of fault classes and diagnostic requirements may evolve over time (e.g., adding new fault categories, changing the label set, or refining decision thresholds). Implementing the classifier in software (or on a reconfigurable platform) facilitates updates and re-training without redesigning silicon, while the always-on AE can remain a compact, stable front-end for event-driven anomaly detection.

Overall, the adopted partitioning represents a system-level trade-off that prioritizes minimal always-on computation near the sensor, reduced data movement, and scalable deployability, while preserving flexibility for the higher-level classifier.

Chapter 5
Case 3: Neural Compensation of Thermal Stress in MEMS Pressure Sensors

V.1 Publications Associated with This Chapter

The case study presented in this chapter builds upon peer–reviewed publications produced during the PhD's broader research on calibration systems for MEMS pressure sensors, conducted in collaboration with STMicroelectronics.

This research led to two journal papers and six conference proceedings. Readers may refer to the following for further details.

V.1.1 Journal Articles

- Paola Vitolo et al. (2025). "Real-time neural network-based thermal stress compensation for pressure sensors in precision localization systems". In: *Microprocessors and Microsystems* 117, p. 105183. ISSN: 0141-9331. DOI: https://doi.org/10.1016/j.micpro.2025.105183

- Danilo Pau et al. (2023). "Tiny Machine Learning Zoo for Long-Term Compensation of Pressure Sensor Drifts". In: *Electronics* 12, 4819.23. ISSN: 2079-9292. DOI: 10.3390/electronics12234819

V.1.2 Conference Proceedings

- Paola Vitolo et al. (2024a). "In-Sensor System for Real-Time Compensation of Thermal Drift in MEMS Pressure Sensors". In: *Proceedings of SIE 2023*. Ed. by Carmine Ciofi and Ernesto Limiti. Cham: Springer Nature Switzerland, pp. 186–191. ISBN: 978-3-031-48711-8

- Paola Vitolo et al. (2024b). "In-Sensor Self-Calibration Circuit of MEMS Pressure Sensors for Accurate Localization". In: *2024 27th Euromicro Conference on Digital System Design (DSD)*, pp. 582–587. DOI: 10.1109/DSD64264.2024.00083

- Paola Vitolo et al. (2023d). "Tiny compensation of pressure drift measurements due to long exposures to high temperatures". In: *2023 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pp. 01–05. DOI: 10.1109/I2MTC53148.2023.10175998
- Gian Domenico Licciardo et al. (2023). "Ultra-Tiny Neural Network for Compensation of Post-soldering Thermal Drift in MEMS Pressure Sensors". In: *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5. DOI: 10.1109/ISCAS46773.2023.10181480
- Danilo Pietro Pau et al. (2024b). "Ultra Tiny Neural Network for Accurate Pressure Sensors Under Multiple Thermal Stresses". In: *2024 IEEE 8th Forum on Research and Technologies for Society and Industry Innovation (RTSI)*, pp. 607–612. DOI: 10.1109/RTSI61910.2024.10761275
- Danilo Pietro Pau et al. (2024a). "Ultra-Tiny Quantized Neural Networks for Piezo Pressure Sensors". In: *2024 Smart Systems Integration Conference and Exhibition (SSI)*, pp. 1–8. DOI: 10.1109/SSI63222.2024.10740508

V.2 Introduction

In recent decades, MEMS pressure sensors have undergone a remarkable evolution in terms of accuracy, integration, and application versatility. Their compact size, low power consumption, and relatively low cost have established them as a key enabling technology across diverse domains, including consumer electronics, automotive systems, biomedical monitoring, and industrial automation (Varshney et al., 2021; Rosário and Dias, 2023). With the increasing demand for pervasive sensing in smart environments, pressure sensors are now frequently integrated into portable and wearable platforms, where continuous monitoring and real-time data acquisition are essential requirements.

A particularly promising field of application is *personal localization* and *emergency response*. MEMS pressure sensors can extract altitude information from precise measurements of atmospheric pressure, enabling the detection of floor levels within buildings, the localization of individuals during evacuation procedures, and assisted navigation for elderly or mobility-impaired users. However, these applications require long-term measurement stability and reliability, which remain a major challenge for MEMS pressure sensors.

Indeed, thermal stresses, which can arise, for example, from reflow soldering processes or prolonged exposure to elevated temperatures, induce residual mechanical strain in the sensor membrane. This strain manifests as drifts in pressure output, typically non-linear and persistent even after environmental conditions have returned to nominal values (Martínez Lahoz et al., 2023; Tran et al., 2019; G. Zhou et al., 2014; Rivera et al., 2007). Such effects undermine the sensor's long-term accuracy, making frequent recalibration necessary throughout its operational lifetime.

Conventional compensation strategies, including one-point or polynomial-based calibration, are attractive for their simplicity but prove inadequate under non-linear or time-varying stress conditions. Moreover, they often rely on external reference measurements or require repeated recalibration steps, which are impractical when sensors

are integrated within complex systems or deployed in inaccessible environments.

To overcome these limitations, recent research has turned to Machine Learning (ML) and Neural Network (NN)-based methods for drift compensation (Najar, 2019; Chang et al., 2020). Thanks to their ability to model highly non-linear relationships, NNs offer a powerful alternative, as also demonstrated by the universal approximation theorem (Kouritzin and Richard, 2024). Several approaches have been proposed, ranging from Extreme Learning Machines (ELMs) (M. Zou et al., 2023), to recurrent NNs (Guo et al., 2023), and Radial Basis Function (RBF) networks optimized via quantum algorithms (Xie et al., 2023). While these techniques significantly improve accuracy when drift and stress occur simultaneously, they generally fail to address long-lasting drifts that persist after the stress has ceased. In addition, their computational and memory demands hinder direct deployment in the resource-constrained environment of MEMS sensors.

In this context, we introduced the **Artificial Intelligence-based Reconfigurable Self-Calibration Unit (AI-ReSCU)**, a novel real-time compensation architecture specifically designed for MEMS pressure sensors (Vitolo et al., 2025). The proposed unit combines a trigger module, responsible for detecting deviations from nominal accuracy, with a compact hardware-implemented NN that estimates and corrects drift errors. The design operates autonomously, without the need for external references or manual intervention. By leveraging resource sharing and clock gating strategies, the AI-ReSCU achieves both compactness and energy efficiency, enabling seamless integration within the sensor circuitry or package.

Experimental results confirm the effectiveness of this approach: the system restores measurement accuracy within ± 0.5 hPa, recovering up to 1.6 hPa of error induced by thermal stress, with a compensation latency of approximately 50 samples in the worst case. The prototype, implemented in STMicroelectronics BCD8 technology, occupies only 0.55 mm^2 and achieves an ultra-low dynamic power consumption of 4.46 nW, demonstrating its suitability for next-generation smart MEMS pressure sensors.

V.3 Related Works

ML and DL techniques have been widely explored for the calibration of pressure sensors subject to environmental changes, aging, and thermal drift. Despite their effectiveness, most of these approaches focus on static calibration and fail to account for the long-term temporal effects of high-temperature exposure, such as the persistent drift observed after soldering processes.

Extreme Learning Machines (ELMs) have emerged as promising candidates for compensating thermal drift in piezoresistive pressure sensors. Reported results show significant accuracy improvements, with temperature coefficients reduced from 2.57%FS to 0.13%FS in the range $[-40, 85]\,^{\circ}\mathrm{C}$ (G. Zhou et al., 2014). ELMs are also attractive due to their short training times, which make them suitable for batch calibration. However, their dependency on high-resource external platforms limits their applicability in low-power embedded systems (G. Zhou et al., 2014). An alternative ELM-based method that exploited non-target parameters, such as temperature and voltage, achieved a mean squared error (MSE) of 0.338 with training times as low as 1.35 s, outperforming both traditional NNs and Support Vector Machines (SVMs) (Chang et

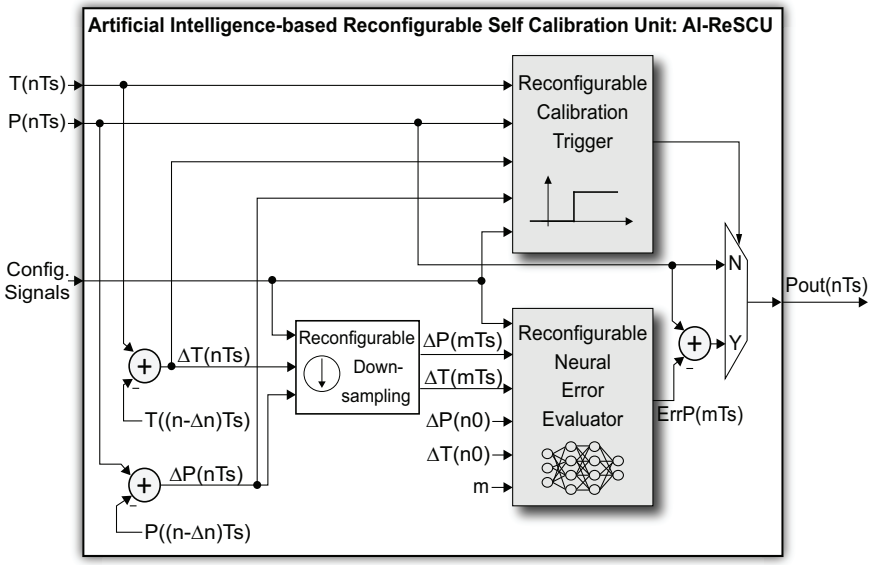

Figure V.1: *Block diagram of the proposed AI-based Reconfigurable Self-Calibration Unit (AI-ReSCU). The architecture integrates the Reconfigurable Calibration Trigger (ReCT), which monitors input signals to detect accuracy drift, and the Reconfigurable Neural Error Evaluator (ReNEE), which computes the correction factor to restore the calibrated pressure output. Extracted from (Vitolo et al., 2025)*

al., 2020).

For capacitive pressure sensors (CPS), innovative electrical-only calibration strategies have been proposed. These methods reduce testing time by over 85% and achieve average accuracy of 1.74 hPa across the 600–1100 hPa range, thus improving cost-effectiveness but without addressing the challenges specific to piezoresistive sensors (Najar, 2019). Other ANN-based approaches have been validated in harsh environments, demonstrating full-scale errors within $\pm 1.5\%$ for nonlinear temperature influences (Patra et al., 2004). More recently, Wavelet Neural Networks (WNNs) have been proposed for temperature compensation, outperforming backpropagation-based NNs with root mean squared errors as low as 3.83×10^{-3} (R. Wu et al., 2023).

Beyond neural models, other notable contributions include adaptive polynomial methods and hardware-based solutions. Polynomial-based compensation achieves accuracies within $\pm 0.068\%$FS while maintaining low hardware complexity (Ali et al., 2020). On the hardware side, a feedforward neural network (FFNN) implemented in a CMOS analog ASIC successfully reduced uncompensated full-scale errors from 9% to 0.1%, using an inverse delayed neuron model (Futane et al., 2010).

To capture the temporal nature of sensor drift, sequential models have also been investigated. Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) have been applied to model drift evolution over time. For example, an LSTM-based approach reduced errors in air pressure sensors from 1.4 kPa to 0.55 kPa, though without improvements in calibration speed (T. Wang et al., 2022). More advanced architectures, such as the Concatenated GRU with Attention (CGDA) model, have

also been proposed, achieving prediction accuracies exceeding 93% in environmental sensors over daily sequences (Chaudhuri et al., 2021).

In summary, while these methods mark significant progress, they fall short in addressing the combined challenges of non-linear temporal drift, ultra-low power consumption, and hardware integration constraints. These limitations underscore the need for compact, reconfigurable, and energy-efficient solutions specifically designed for real-time, long-term correction of sensor drift in MEMS devices.

V.4 AI-ReSCU: AI-based Reconfigurable Self-Calibration Unit

The proposed AI-ReSCU is designed to autonomously compensate for accuracy drifts in MEMS pressure sensors induced by temporal thermal stress. As depicted in Fig. V.1, the system is organized around two key modules: the *Reconfigurable Calibration Trigger* (ReCT) and the *Reconfigurable Neural Error Evaluator* (ReNEE).

The **ReCT** module continuously monitors the sensor's input signals, namely, the temperature $T(nT_s)$, the pressure $P(nT_s)$, and their temporal variations $\Delta T(nT_s)$ and $\Delta P(nT_s)$. These signals are sampled at discrete instants nT_s, where T_s is the reciprocal of the sensor's ODR. By evaluating whether the input signals, alone or in combination, exceed configurable thresholds, the ReCT determines when recalibration is required and, if necessary, activates the compensation process. All trigger configurations are listed in Table V.1. These modes are selectable via dedicated configuration signals, which also allow the tuning of threshold values. To ensure robustness against noise and spurious variations, an adjustable delay mechanism prevents false triggers.

The **ReNEE** module is responsible for computing the correction factor needed to restore sensor accuracy. It relies on a compact Neural Network (NN), which processes the temporal variations $\Delta T(mT_s)$ and $\Delta P(mT_s)$ to estimate the error $ErrP(mT_s)$ and generate the corrected pressure output $P_{out}(nT_s)$. Thanks to its reconfigurable design, the ReNEE can adapt to diverse operating conditions and sensor characteristics.

To reduce computational overhead, the inputs of the ReNEE are downsampled prior to processing. This is feasible because the dynamics of thermal drift are typically much slower than the sensor sampling frequency. The downsampling factor is configurable, thus enabling trade-offs between responsiveness and energy efficiency. Moreover, the activation period of the ReNEE can be tuned: once the drift returns within acceptable limits, the module automatically deactivates to minimize power con-

Table V.1: *List of Supported Trigger Configurations. Extracted from (Vitolo et al., 2025).*

Trigger Configuration	Description
00	P >Pmax or T>Tmax or ΔP>ΔPmax or ΔT>ΔTmax
01	(P>Pmax and T>Tmax) or (ΔP>ΔPmax and ΔT>Δtmax)
10	(P>Pmax and ΔP>ΔPmax) or (T>Tmax and ΔT>Δtmax)
11	P>Pmax and T>Tmax and ΔP>ΔPmax and ΔT>Δtmax

ΔP=P(nTs)-P(nTs-Δt); ΔT=T(nTs)-T(nTs-Δt);
Pmax, Tmax, ΔPmax, and ΔTmax are configurable threshold values;
Δt=configurable delay time; Ts=sampling period.

Table V.2: *List of Supported Neural Network Configurations. Extracted from (Vitolo et al., 2025).*

Neural Network Configuration	Description
Layer type	1D Convolutional or Fully Connected
Numbers of Hidden Layers	[1:8]
Number of Neurons per Layer	[1:20]
Activation Function	Linear or ReLU

sumption.

Finally, the AI-ReSCU supports multiple neural network configurations, as summarized in Table V.2. The NN can be customized in terms of:

- number of hidden layers (1–8),
- number of neurons per layer (1–20),
- layer type (Fully Connected or 1D Convolutional),
- activation function (Linear or ReLU, defined in Eq. V.1).

$$ReLU(x) = \begin{cases} 0 & for \quad x < 0 \\ x & for \quad x \geq 0 \end{cases} \tag{V.1}$$

V.5 AI-ReSCU Hardware Architecture

The hardware design of the proposed AI-ReSCU prioritizes ultra-low power consumption and minimal silicon area to meet the stringent constraints of in-sensor computing, the need to integrate extra logic within the sensor readout chain under sub-mm^2 area budgets, μW–mW average-power envelopes for (quasi) always-on operation, and very limited on-chip memory, while minimizing data movement and avoiding self-heating effects that could degrade sensor performance. hese constraints ensure that the footprint and power dissipation of the unit remain compatible with the circuitry already integrated within MEMS pressure sensors.

To optimize resource usage while preserving real-time operation, the AI-ReSCU adopts an *iterative*, highly resource-shared microarchitecture, following the strategy in (**vitolo˙tcas2**; **2021˙icecs˙vitolo**). Although iterative reuse slightly increases processing latency, it remains fully compatible with typical sensor Output Data Rates (ODR) of 1–200 Hz (i.e., sampling periods down to 5 ms). Power efficiency is further enhanced through aggressive clock gating, which disables inactive modules across the inference cycle.

Figure V.2 shows the top-level hardware organization. The architecture instantiates the ReCT and the ReNEE, mirroring the conceptual scheme of Fig. V.1. A bank of registers stores temporal samples and configuration parameters (e.g., thresholds, downsampling factor, compensation interval/period, and NN hyperparameters), while

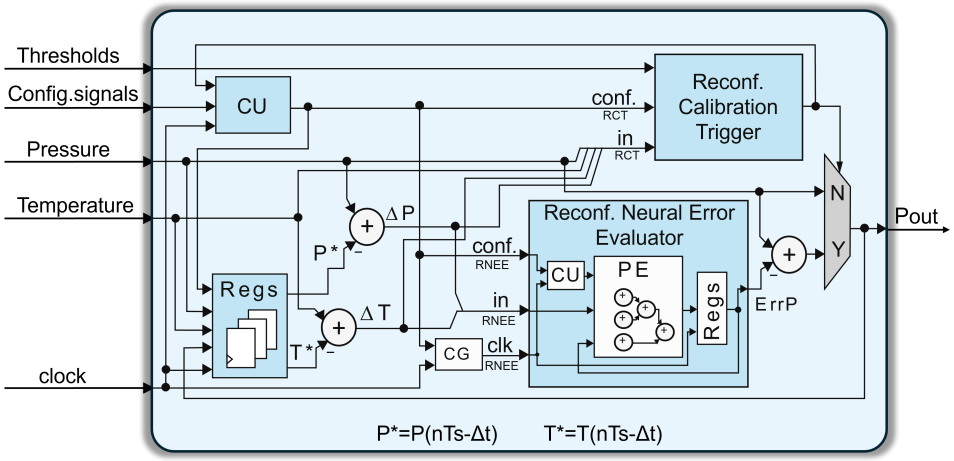

Figure V.2: *Hardware architecture of the AI-ReSCU. Inputs include configurable thresholds ($P_{\max}$, $T_{\max}$, $\Delta P_{\max}$, $\Delta T_{\max}$) and configuration signals (trigger mode, delay times, downsampling factor, compensation interval/period, and NN hyperparameters). An FSM coordinates ReCT and ReNEE and controls clock gating for ultra-low-power operation. Extracted from (Vitolo et al., 2025).*

a Finite-State Machine (FSM) orchestrates dataflow, module activation, and clock gating. A lightweight configuration interface (e.g., SPI) loads network parameters and runtime settings at startup.

The ReCT module (Fig. V.3) continuously monitors $P(nT_s)$, $T(nT_s)$ and their temporal variations $\Delta P(nT_s)$, $\Delta T(nT_s)$ to detect uncalibrated conditions. A 4-input selection network implements the trigger modes in Table V.1. To mitigate false positives caused by noise or transients, ReCT embeds a configurable delay/hysteresis stage. When a drift condition is asserted, ReCT awakens ReNEE for compensation; otherwise, only the ultra-light ReCT remains active.

The configurable delay/hysteresis stage can be interpreted as a delay-based debouncing mechanism that stabilizes the trigger decision in the presence of sensor noise, quantization effects, and short transients. In practice, a threshold-crossing event does not immediately activate the compensation pipeline: the drift condition must persist for a programmable delay interval before being confirmed. Symmetrically, once the drift flag is asserted, the trigger is released only after the monitored quantities return within a safe region for a sufficient persistence interval, thereby introducing hysteresis and preventing rapid toggling (chattering) around the decision thresholds. This mechanism reduces spurious activations of ReNEE and improves both robustness and energy efficiency, since the heavier NN-based compensation is executed only when a drift condition is stable rather than momentary.

The delay interval is configurable (Table V.1), allowing the debouncing aggressiveness to be tuned to the expected sensor dynamics and noise level.

The ReNEE module executes the neural error estimator with a single reusable Processing Element (PE), which iteratively performs all MAC operations across layers. This maximizes resource sharing and minimizes memory. The PE features a

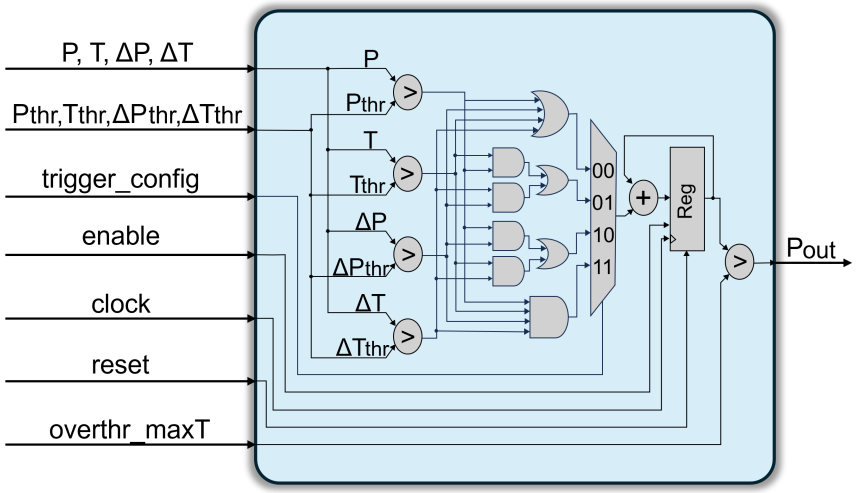

Figure V.3: *Block diagram of the ReCT trigger module. A configurable multi-input comparator evaluates the trigger modes of Table V.1 on P, T, ΔP, ΔT, with programmable thresholds and delay-based debouncing. Extracted from (Vitolo et al., 2025).*

three-level adder tree operating in fixed-point 24-bit (12.12) precision to guarantee numerical stability.

To further reduce memory and arithmetic complexity, the NN adopts *partial quantization* (**2021˙icecs˙vitolo**; Antonio De Vita et al., 2020; Vitolo et al., 2022a; Vitolo et al., 2023b): all weights are binarized to $\{-1, +1\}$ (1 bit), while activations are kept at 24 bits. Under this regime, MACs collapse to signed add/sub operations, eliminating multipliers and yielding substantial savings in both area and dynamic power.

Because thermal drift evolves slowly relative to the ODR, the ReNEE input stream is *downsampled* by a configurable factor prior to inference. This reduces compute and allows duty cycling of the NN engine. The current error estimate is written to a dedicated register and applied to compute the corrected pressure $P_{\text{out}}(nT_s)$. After each inference, ReNEE is gated off and remains idle for a programmable *compensation interval*. This process repeats for a configurable *compensation period*; when the drift falls back within nominal thresholds, ReNEE remains off until ReCT re-triggers compensation.

Overall, the combination of (i) always-on lightweight triggering, (ii) iterative single-PE inference, (iii) binarized weights with fixed-point activations, (iv) aggressive clock gating, and (v) input downsampling enables real-time operation within the nanowatt power envelope and sub-mm^2 area typical of in-sensor deployments.

V.6 Case Studies and Dataset Preparation

The performance of the proposed AI-ReSCU was validated on two experimental case studies using pressure and temperature data acquired from ST piezoresistive MEMS pressure sensors:

- **Reflow soldering:** the sensors were stressed according to the time–temperature

profile specified by the IPC/JEDEC J-STD-020C standard (IPC/JEDEC, 2004);

- **High-temperature exposure:** the sensors were maintained at $100\,^{\circ}\mathrm{C}$ for two hours, beyond their nominal operating range.

These scenarios were selected to cover contrasting conditions. Reflow soldering induces short but intense thermal cycles, with peaks up to $260\,^{\circ}\mathrm{C}$ lasting 10–40 s (IPC/JEDEC, 2004). Conversely, the high-temperature exposure applies a lower temperature for a prolonged period (two hours at $100\,^{\circ}\mathrm{C}$).

All experiments used the *LPS22HH*, an ST MEMS pressure sensor designed as a digital barometer (STMicroelectronics, 2019). The device measures absolute pressure in the 260–1260 hPa range and operates from -40 to $85\,^{\circ}\mathrm{C}$. The embedded temperature and pressure outputs feature 24-bit and 16-bit resolutions, respectively.

Data were collected at the ST MEMS Sensor Characterization and Measurement Laboratory. The reflow-soldering campaign involved 80 LPS22HH units, whereas the high-temperature exposure involved 6 units. Accuracy was computed by comparing stressed devices against an unstressed reference sensor.

As illustrated in Fig. V.4 and Fig. V.5, thermal stress produces a long-lasting accuracy drift that persists over several days. In the high-temperature case, the variability is markedly higher and the dataset clusters into three distinct curve families (Fig. V.5); the maximum standard deviation reaches 49.92 Pa, more than twice that observed in the soldering scenario.

For neural-network training, datasets were obtained by selecting samples within $\pm 2.5\sigma$ from the experimental distributions. Specifically, we generated 100 time–pressure curves for the soldering scenario and 100 curves for each of the three classes identified in the high-temperature exposure case.

V.7 Compensation Results

An extensive model ablation study related to the two case studies was previously reported in (D. Pau et al., 2023). In that work, 53 different Machine Learning (ML) and Deep Learning (DL) models were developed and analyzed, including Temporal Convolutional Networks (TCN), Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) networks, Gated Recurrent Units (GRU), Legendre Memory Units (LMU), Random Forest Regressors (RFR), and Support Vector Regressors (SVR).

Among the explored solutions, the Neural Network (NN) architecture that offered the best trade-off between model complexity and accuracy consists of four layers:

- one fully connected layer with 9 neurons;
- two 1D convolutional layers (kernel size = 3, stride = 1, valid padding);
- one fully connected layer with a single neuron, outputting the compensation term ΔP.

All hidden layers employ the ReLU activation function. Activations are quantized using a 24-bit fixed-point representation, while weights are binarized to 1 bit.

This compact configuration results in only 101 parameters. The model was designed, quantized, and trained in Python using TensorFlow and QKeras (Abadi et al.,

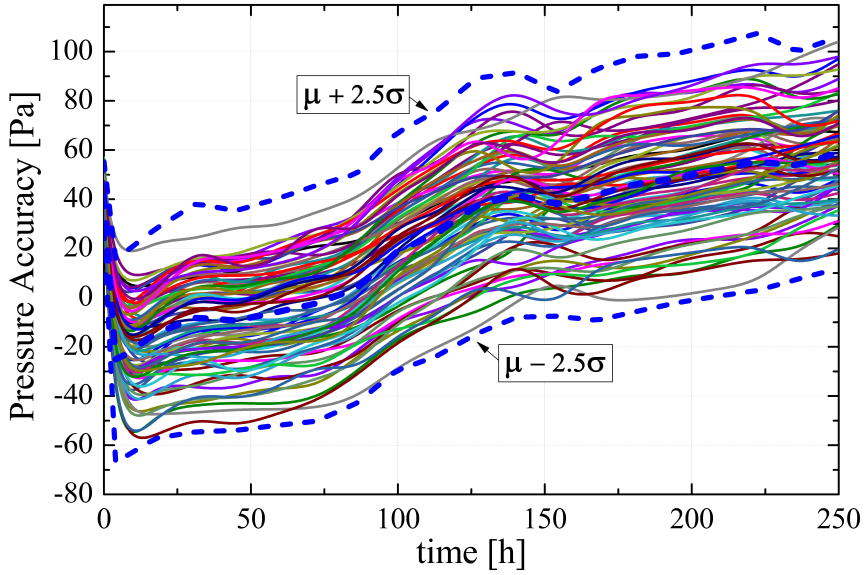

Figure V.4: *Pressure accuracy for 80 LPS22HH devices subjected to reflow thermal stress following IPC/JEDEC J-STD-020C (IPC/JEDEC, 2004). Extracted from (Vitolo et al., 2025).*

2016; Coelho et al., 2021). The dataset was split into 80% training, 10% validation, and 10% test sets. Training was performed for 50 epochs using the ADAM optimizer, with Mean Squared Error (MSE) as the loss function. Both MSE and Mean Absolute Error (MAE) were employed as evaluation metrics.

Offline evaluation on the test set yielded the following results. In the soldering scenario, the NN achieved an MSE of 446 Pa and an MAE of 17.8 Pa. In the high-temperature scenario, three distinct curve classes were identified:

- Class 1: MSE = 0.8 Pa, MAE = 7.3 Pa;
- Class 2: MSE = 3.9 Pa, MAE = 17.9 Pa;
- Class 3: MSE = 1.4 Pa, MAE = 10.3 Pa.

These results confirm that the NN is capable of compensating both offset and non-linearity errors by learning systematic deviations from nominal behavior. Across all cases, the proposed unit successfully restored sensor accuracy within ± 0.5 hPa. Accuracy improvements reached 76% for the soldering scenario and 35%, 67%, and 91% for the three classes of the high-temperature exposure scenario, respectively.

A qualitative example is shown in Fig. V.6, which compares the raw drifted signal of a soldered device with the corrected output obtained from the NN compensation. The compensated curve closely follows the nominal reference, demonstrating effective correction of both baseline offsets and nonlinear drift components. This graphical evidence reinforces the quantitative findings reported above.

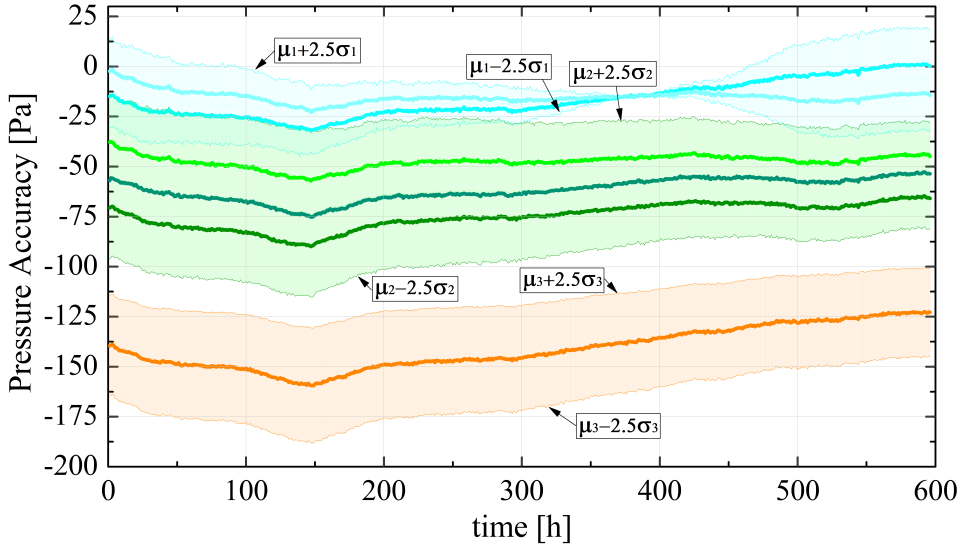

Figure V.5: *Pressure accuracy for 6 LPS22HH devices after two hours at* 100 °C. *Extracted from (Vitolo et al., 2025).*

V.8 Hardware Performance

To assess the performance of the AI-ReSCU, we designed a test chip in STMicroelectronics BCD8 technology and integrated a Serial Peripheral Interface (SPI) wrapper for configuration and data loading. A FSM supervises the operating flow during evaluation, coordinating startup, parameter loading, and runtime inference with aggressive clock gating.

Figures V.7, V.8, and V.9 report post-implementation simulations that illustrate the system behavior across the main operating phases: initialization, configuration loading, and runtime operation with energy-saving mechanisms.

Figure V.7 shows the overall operational sequence, spanning both startup and runtime. After power-on/reset, the control FSM initiates the loading of all configuration data: neural-network weights and thresholds, trigger settings, and compensation parameters (interval and period). These values are serialized and stored into internal registers via the SPI interface. Throughout this phase the gated NN clock (`NN_clk`) remains low, ensuring the computational core is idle until inference is required, thereby minimizing unnecessary energy consumption.

A detailed view of the startup is provided in Fig. V.8, which zooms into the register-loading activity. The complete initialization requires precisely 3206 system clock cycles.

Once initialization is complete, the system enters its runtime mode (Fig. V.9). Pressure and temperature samples are acquired periodically at intervals determined by the sensor Output Data Rate (ODR). The ReNEE module–which embeds the NN inference engine–is enabled only when a new compensation must be computed. As highlighted in the waveform, a full NN inference takes 593 system clock cycles; imme-

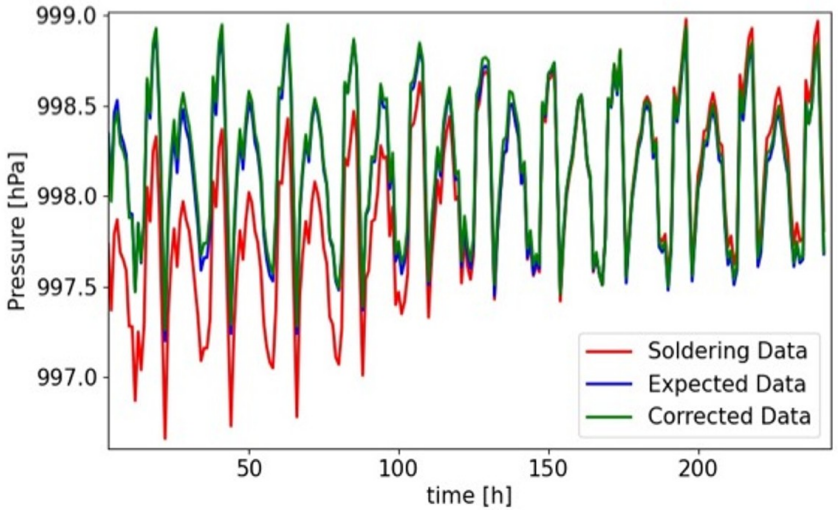

Figure V.6: *Compensation results for the soldering scenario. The raw drifted output (red) is compared against the expected nominal reference (blue) and the corrected signal obtained through NN-based compensation (green). The proposed method effectively suppresses both offset and nonlinear drift, restoring nominal sensor accuracy. Extracted from (Vitolo et al., 2025).*

diately after, the NN is clock-gated back to idle. This duty cycling markedly reduces average power while preserving real-time operation.

The dynamic response (i.e., time to restore acceptable accuracy after an error is detected) was also evaluated. In the worst case, with ODR = 1 Hz and a 12 Hz system clock (minimum for real-time execution), a compensation requires 593 clock cycles, corresponding to $\approx$49.4 s $(593/12)$ or about 50 input samples between drift detection and recovery of nominal performance.

The architecture is highly compact thanks to resource sharing and iterative processing. The total silicon footprint is 0.55 mm^2, with ReNEE accounting for $\sim$94% of the area.

Power is dominated by leakage, while dynamic consumption depends on clock frequency, ODR, and NN activity (clock gating). At $V_{\mathrm{DD}} = 1.8$ V:

- **ODR = 1 Hz.** When ReNEE is idle, total power is 2.339 μW (dynamic 4.456 nW). When ReNEE is active, total rises to 2.473 μW (dynamic 8.586 nW). Leakage accounts for 99.81% (idle) and 99.65% (active).
- **ODR = 200 Hz.** With ReNEE idle, total power is 2.938 μW (dynamic 0.604 μW). When active, total is 4.027 μW (dynamic 1.562 μW). Leakage remains predominant: 79.44% (idle) and 61.21% (active).

These results confirm that (i) leakage is the main contributor to total power, and (ii) the combination of input downsampling and aggressive duty cycling of the NN engine keeps dynamic power in the nW–μW range across the full ODR span, meeting the ultra-low-power envelope required for in-sensor integration.

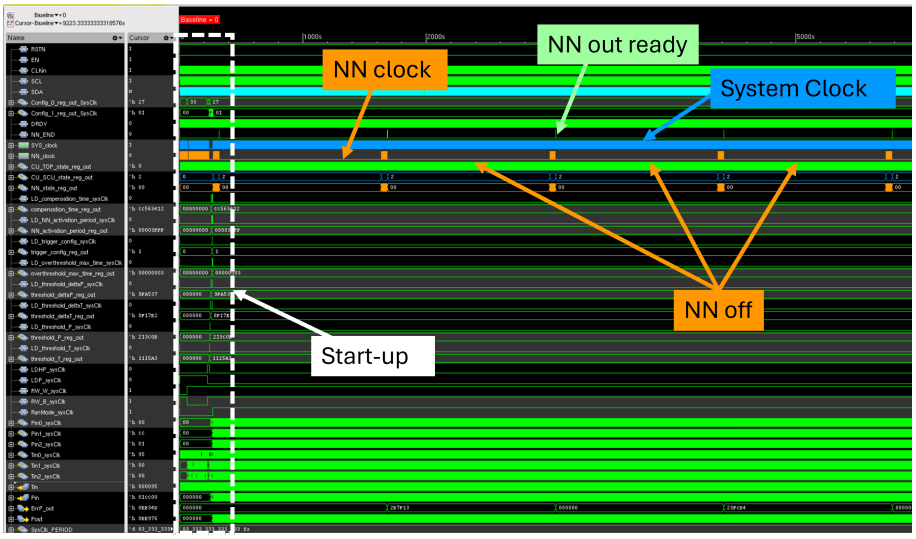

Figure V.7: *Overall behavior of the AI-ReSCU. After startup and parameter loading via SPI (network parameters, thresholds, trigger configuration, compensation interval/period), the system enters runtime. Samples arrive every T_s (inverse of ODR); the corrected pressure is produced at the output. The NN engine is activated only for the time needed to compute the compensation, then gated off until the next compensation interval elapses.*

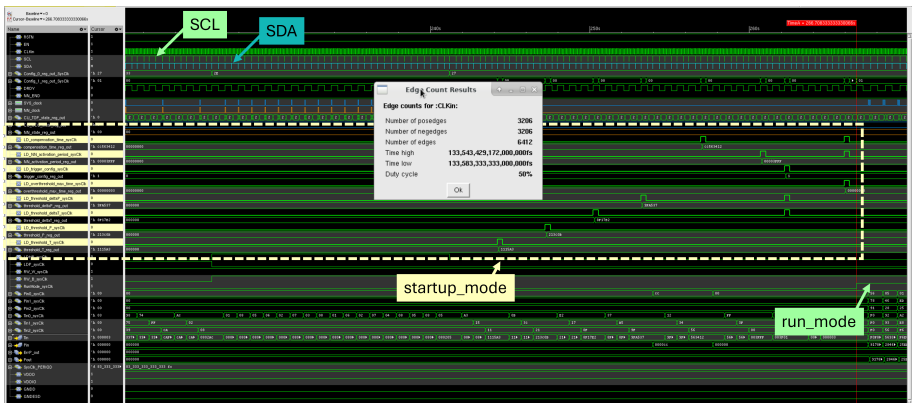

Figure V.8: *Startup waveforms. Initialization and loading of NN parameters, thresholds, trigger configuration, and compensation timing values occur via SPI. The startup phase completes in 3206 system clock cycles; during this phase, the NN clock remains gated.*

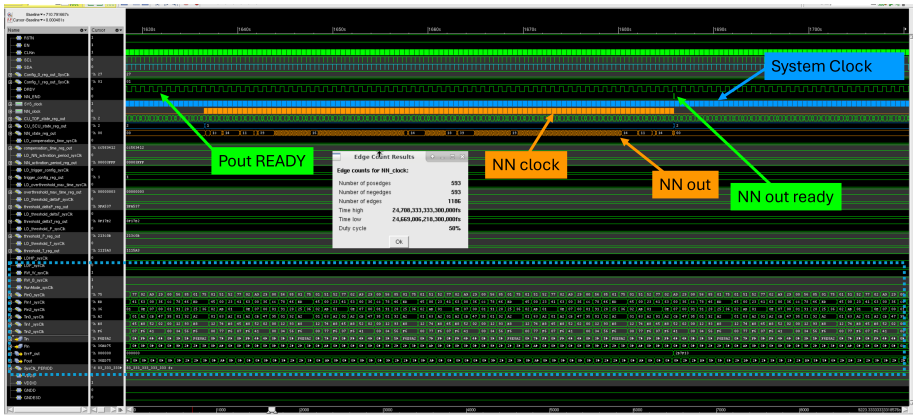

Figure V.9: *Runtime waveforms. Pressure and temperature are sampled every T_s; upon request, ReNEE performs inference in 593 cycles and is then clock-gated to idle. Duty cycling of the NN engine minimizes dynamic power while preserving real-time correction.*

Chapter 6
Discussion and Perspectives

VI.1 Integration of Contributions

This dissertation investigated how to realize neural processing directly within or next to sensors under extreme energy and silicon constraints. Across three application domains—audio front-end for keyword spotting, vibration-based predictive maintenance, and thermal-stress compensation in MEMS pressure sensors—we adopted a software–hardware co-design methodology that couples model design with tailored digital hardware accelerators. The results demonstrate that ISC can deliver competitive accuracy with orders-of-magnitude reductions in hardware resources and energy, provided that networks and hardware are co-constrained from the outset.

The key unifying principles of this dissertation are:

- *software–hardware co-design with early constraint propagation* and iterative feedback from synthesis/implementation back to modeling, so the NN is shaped by area/power/timing limits from the start; *constraints-first, iterate, converge*.

- *resource sharing and serialized compute* (single reusable PE + FSM control), maximizing on-chip reuse of inputs, weights, and partial sums, to fit tiny on-chip memories and keep leakage low, accepting modest latency when compatible with sensor ODRs.

- *custom quantization*, including low-bit fixed-point and selective binarization, applied already during training to improve model accuracy, then implemented in hardware to minimize memory usage and power;

- *event-triggered self-activation and deep sleep of processing blocks*, with compute split where appropriate (e.g., an always-on in-sensor stage with an on-demand MCU head, or partitioned ASIC domains controlled by clock gating so that functional blocks such as the neural calibration engine are enabled only when required), allowing compute units to wake autonomously when sensor-detected conditions demand processing and keeping overall power consumption to a minimum.

These principles were validated on FPGA and synthesized in CMOS standard-cell flows, and, when fabrication or full-system prototyping was available, verified in real-

istic streaming settings, yielding competitive accuracy with orders-of-magnitude savings in energy and silicon area.

Moreover, applied across the three case studies that differ in application domain, signal modality, and time constants yet share the same ISC bottleneck, they demonstrate clear cross-domain portability.

- *Audio front-end*: a tiny 1D-CNN fuses filtering and decimation (overall $R{=}128$ from a 2.048 MHz 1-bit PDM stream to 16 kHz 8-bit PCM), replacing the CIC+FIR chain; the top-level hardware reuses a single PE across layers under a finite-state controller, staging weights and partial sums in small on-chip buffers to reduce area and power. The design achieves $\sim$48 dB SNR and $\sim$89% KWS accuracy with a synthesized footprint of $\sim$0.09 mm^2 and $\sim$128 μW/MHz in 130 nm CMOS.

- *Predictive maintenance*: a hybrid, event-driven pipeline combines an always-on, partially binarized in-sensor AE for anomaly detection (99.61% detection) with an MCU classifier that wakes only on events (up to 94.83% across 9 classes), supporting ODRs up to 365 kHz; the AE prototype shows $\sim$333 μW/MHz dynamic on FPGA and, in 65 nm, $\sim$0.49 mm^2 and $\sim$138.6 μW/MHz dynamic.

- *Thermal compensation for MEMS pressure sensors*: the proposed AI-ReSCU couples a reconfigurable trigger with a compact NN corrector that runs for just the time required to compute the compensation, after which the engine returns to deep idle through clock gating, delivering sub-20 μs inference while fitting tight in-sensor budgets ($\sim$4.46 nW and $\sim$0.55 mm^2).

Although each case-study chapter reports power, area, and timing results compared with the state of the art in detail, it is useful to provide a consolidated *budget-style* summary that makes the ISC constraints explicit and enables quick comparison. Across the three case studies, the design targets were defined by tight sensor-integration envelopes: a *sub-mm*^{2} silicon footprint for the added digital core, *sub-mW* power for always-on or quasi-always-on operation, and *deterministic real-time execution* compatible with the sensor output data rate (ODR) or sampling period. Table VI.1 summarizes the achieved operating points and derives a simple energy metric per output sample, per detector window, or per compensation event using values already reported in the dedicated chapters.

Table VI.1 highlights three different operating regimes enabled by the same constraints-first co-design principles: (i) *always-on streaming* with deterministic per-sample budgets (Case 1), (ii) *event-driven monitoring* where classification is paid only upon anomalies (Case 2), and (iii) *aggressive duty cycling* where the neural core is clock-gated for most of the time and energy is best interpreted per compensation event (Case 3).

Quantitatively, as for the Case 1, at the target microphone ODR, the proposed neural PDM-to-PCM converter achieves a sub-mm^2 footprint and sub-mW power, enabling a derived energy cost on the order of tens of nJ per PCM sample while meeting the strict real-time output period imposed by the 16 kHz stream. In Case 2, the budget must be interpreted in terms of *average* energy: the always-on AE dominates the continuous cost, whereas the CNN head is activated sporadically, so classification energy is incurred only upon detected anomalies; consequently, the relevant time scale

is set by the detector window length (and thus by W/ODR), rather than by a fixed per-sample period. Case 3 represents the most extreme duty-cycled regime: the NN engine is invoked intermittently and quickly returned to idle through clock gating; in this setting, the dynamic energy per compensation event is small, while the total energy can become leakage-dominated at very low ODR, which motivates focusing on event-level budgeting rather than throughput-centric metrics.

Overall, the table makes explicit how the same design principles—early constraint propagation, aggressive resource sharing, and activity-aware activation—map to different optimal operating points depending on whether the system is streaming, event-driven, or intermittently triggered.

The three case studies were selected because they originate from concrete industrial requirements and provide three representative, yet heterogeneous, ISC scenarios in which an *always-on* sensing node must extract actionable information under tight integration constraints. Rather than targeting a single application domain, the thesis covers different sensor types (microphone, accelerometer, pressure sensor), different signal time constants (from high-rate streaming to slow thermal dynamics), and different application objectives (audio front-end for KWS, vibration-based predictive maintenance, and sensor self-calibration). This diversity offered the opportunity to apply the same constraints-first hardware–software co-design methodology across markedly different sensing modalities and operating regimes, thereby testing its generality and transferability. While the selected case studies already span a broad range of practical smart-sensor needs, future work can extend the validation to additional sensing domains and applications as new industrial requirements arise.

In addition to the three application-driven ISC case studies, this thesis reports an *exploratory* investigation on AI-assisted hardware development using Large Language Models (LLMs). The goal is to address a key bottleneck of the proposed methodology—the high degree of hardware customization—by targeting one of its most time-consuming and expertise-intensive stages: the development of synthesizable HDL modules and their verification environments. This study is not intended to introduce a fourth ISC case study, but to evaluate whether LLMs can act as productivity aids for drafting HDL and verification artifacts under designer supervision and rigorous validation.

From the perspective of the constraints-first co-design methodology proposed in Chapter II, AI-assisted approaches can be integrated as supporting tools at multiple points in the flow. For example, starting from explicit requirements and a fixed, quantized model specification, LLMs can help generate RTL skeletons consistent with a given micro-architecture, produce testbench templates and assertions, assist documentation, and accelerate iterative refinement by summarizing synthesis and simulation feedback. Importantly, correctness and deployability must remain guaranteed by conventional design practices: functional verification against a reference model, timing/power closure through EDA tools, and reproducibility through transparent auditing of prompts and design iterations.

Overall, this study suggests that LLMs can support selected non-recurring engineering tasks in the hardware-development workflow (e.g., HDL design and verification artifacts), provided that correctness and deployability remain guaranteed by constraint-driven design choices and rigorous verification.

VI.2 Limitations and Open Challenges

While the results across audio, vibration, and pressure sensing are promising, several limitations emerged that outline where in-sensor computing must mature before broad adoption. First, robustness and transferability remain challenging once systems leave curated datasets and controlled labs. Variations in enclosures, mounting, temperature histories, and manufacturing tolerances can shift data distributions and degrade performance; yet, field retraining is constrained by tight energy, memory, and time budgets. This calls for carefully designed calibration policies, lightweight on-device adaptation, and data-collection campaigns that truly span the operational envelope.

A second limitation concerns the maturity of end-to-end toolchains. Despite advances in quantization-aware training and integer-only flows, moving from high-level models to consistent, synthesizable RTL still requires bespoke glue: operator substitutions, fixed-point range alignment, memory mapping, and deterministic test-vector generation. Numeric mismatches between software models and hardware (e.g., saturation, rounding, overflow) can surface late in the flow, and there is no widely adopted intermediate representation that carries fixed-point semantics, memory plans, and control protocols across ML and EDA stacks.

Scalability is another open issue. The designs intentionally favor serialized, resource-shared compute to minimize area and leakage, but higher-ODR sensors or arrayed modalities will demand selective parallelism and deeper buffering. That, in turn, stresses interconnects and memory budgets and can erode energy proportionality if not paired with hierarchical clock/voltage domains and precise wake-up policies. Moreover, technology choices involve trade-offs between leakage and maximum frequency that must be revisited per application and node.

In this thesis, AI-assisted RTL generation is discussed as an exploratory, complementary direction (reported separately from the three ISC case studies) of the proposed methodology. When generative AI assists RTL creation, verification becomes the bottleneck rather than coding. LLMs accelerate scaffolding and documentation, but they do not guarantee correctness or coverage. Safety at scale requires property-driven design, constrained-random verification, and systematic equivalence checks between quantized models and netlists, alongside security reviews for unintended behaviors at interfaces. Overall, these challenges point to a coordinated agenda that couples data-centric design with standardized tooling and rigorous verification, so that tiny, energy-proportional intelligence at the sensor can be deployed with confidence.

VI.3 Future Directions

Building on the three case studies, several directions can further mature in-sensor computing toward robust, scalable deployments. First, a data-centric agenda is essential: datasets should reflect the full operating envelope of each sensor (enclosures, mounting, temperature/humidity cycles, aging), with targeted augmentation for rare events and long-tail conditions. Lightweight telemetry hooks and shift detectors embedded at the edge can flag distribution drift, triggering safe re-calibration or model refresh. Where privacy or bandwidth is constrained, federated or on-device strategies for collecting statistics and updating calibration parameters can maintain performance without exporting raw data.

At the architectural level, future designs should generalize the resource-shared micro-architectures introduced here to higher-ODR and arrayed modalities. Selective parallelism, deeper but carefully banked buffers, and hierarchical clock/voltage domains can preserve energy proportionality while meeting tighter latency targets. Multi-rate pipelines and event-coded representations (e.g., sparse updates, anomaly-driven bursts) can further amortize data movement. Extending the audio front-end to small microphone arrays with streaming beamforming, scaling vibration monitoring to multi-axis or multi-sensor nodes, and broadening pressure-sensor compensation to coupled environmental variables (e.g., humidity) are concrete steps in this direction.

Mixed-signal and in-memory opportunities deserve systematic exploration. Charge- or time-domain primitives can implement ultra-low-energy pre-filtering or encoding ahead of the tiny digital correctors demonstrated here. Embedded non-volatile memories (FRAM/MRAM/ReRAM) offer near-zero standby leakage for parameters and, where variability and endurance permit, limited in-memory MAC for selected layers. Co-designing these elements with robust digital fallbacks and self-check calibration loops can hedge against device non-idealities.

For adaptation in the field, on-device learning must be pursued under strict, auditable budgets. Bounded-cost optimizers, sparse or episodic updates scheduled by well-defined triggers, and rollback-safe mechanisms (A/B parameter banks, checksummed snapshots) can deliver slow drift compensation without compromising availability. Integer-only fine-tuning paths and quantization-preserving updates will keep the learning loop consistent with the deployed hardware.

Tooling and verification should converge toward standardized interfaces that carry fixed-point semantics, memory plans, and I/O protocols from ML frameworks to RTL and EDA. Golden-reference generators that emit deterministic test vectors, property libraries for saturation/overflow and latency contracts, and coverage metrics tailored to streaming dataflows will reduce late-stage mismatches. Security and resilience also warrant attention: fault injection, adversarial anomaly scenarios, and interface fuzzing should be part of the acceptance gate for safety-critical nodes.

Finally, AI-in-the-loop EDA can evolve from ad-hoc prompting to instrumented assistance. Within the constraints-first flow of Chapter II, such assistance would naturally operate after requirements and numerical formats are fixed, supporting RTL drafting and verification generation while preserving constraint-driven acceptance criteria. Beyond scaffolding modules and testbenches, language models can propose assertions from interface contracts, summarize synthesis and timing logs into actionable micro-architecture hints, and auto-generate scenario-driven regression suites. Equally important is auditability: prompt/version tracking, rationale extraction, and structured reviews are needed to understand and validate LLM-generated choices before integration into safety- or reliability-critical hardware flows. Guardrails—linting, formal checks, and human sign-off—remain central, but with these in place, AI assistance can shorten the concept-to-prototype path and broaden access to deployable, energy-proportional intelligence at the sensor edge.

Table VI.1: *Summary of area, power and latency across the three case studies. Energy figures marked as "derived" are obtained from reported power and the relevant time scale (sample/window/event).*

	Case 1: Audio front-end (KWS)	**Case 2: Predictive maintenance**	**Case 3: AI-ReSCU (pressure)**
ISC budget target (typ.)	Sub-mm^2 added digital core and sub-mW power, with real-time behavior compatible with sensor ODR / sampling period.		
Operating point	PDM ODR = 2.048 MHz; output PCM = 16 kHz (8-bit)	AE always-on; reported at ODR = 20 kHz (ASIC operating point)	ODR = 1–200 Hz (reported figures shown for ODR = 1 Hz)
Area [mm^2]	0.086 (130 nm CMOS)	0.49 (65 nm LP HVT)	0.55 (BCD8 testchip)
Power (reported)	$P_{\mathrm{dyn}} = 837\,\mu$W @ ODR=2.048 MHz (CIC baseline: 2600 μW)	$P_{\mathrm{tot}} = 341\,\mu$W @ ODR=20 kHz (classifier on-demand)	$P_{\mathrm{tot}} = 2.339\,\mu$W idle; 2.473 μW active (dynamic: 4.456–8.586 nW @ ODR=1 Hz)
Latency metric (reported)	Deterministic output period: $T_{\mathrm{PCM}} = 1/16\,\mathrm{kHz} = 62.5\,\mu$s	Detection time scale set by windowing: $W = 24$ samples $\Rightarrow T_W = W/\mathrm{ODR}$ (e.g., 1.2 ms @ 20 kHz)	Inference latency: 593 clock cycles (worst case: $\sim$49.4 s @ 12 Hz clock)
Energy metric	*Derived:* $E_{\mathrm{PCM}} = P_{\mathrm{dyn}} \cdot T_{\mathrm{PCM}} \approx$ 52.3 nJ/sample (baseline $\approx$ 162.5 nJ/sample)	*Derived:* $E_W = P_{\mathrm{tot}} \cdot T_W \approx$ 409 nJ/window (at ODR=20 kHz, T_W=1.2 ms)	*Derived (dynamic):* $E_{\mathrm{dyn}} \approx P_{\mathrm{dyn,act}} \cdot T_{\mathrm{inf}} \approx$ 0.424 μJ (worst-case; total energy leakage-dominated)

Conclusions

This dissertation has demonstrated that neural network inference can be achieved within or alongside sensors under extreme resource constraints by adopting a rigorous hardware–software co-design approach. The research addressed the challenge of in-sensor computing by jointly optimizing tiny neural models and custom digital accelerators, ensuring that even under millimeter-scale silicon areas and microwatt power budgets, data-driven intelligence can reside at the edge. Through three collaborative case studies with STMicroelectronics, the feasibility and benefits of this approach were validated across diverse sensing domains, highlighting both the technical innovations and the broader implications for next-generation smart sensing systems.

In the first case study, a keyword spotting (KWS) application for digital MEMS microphones, the traditional multi-stage PDM-to-PCM filtering chain was replaced with a compact neural network. This neural front-end performed on-sensor audio decimation and feature extraction, enabling an integrated always-on KWS system that achieves competitive accuracy with 48 dB SNR and 89% classification accuracy while adding minimal overhead (approximately 0.09 mm^2 of silicon and consuming only 128 μW/MHz in 130 nm CMOS). The second case study tackled vibration-based predictive maintenance, combining anomaly detection and classification in a two-tier architecture. A partially binarized autoencoder continuously monitors accelerometer data for anomalies, and upon detection, activates a lightweight CNN to classify the fault. This event-driven scheme maintains high diagnostic accuracy for machine condition monitoring while keeping the average power draw extremely low by only expending energy on classification when needed. The third case study focused on thermal drift compensation in MEMS pressure sensors, where a neural network was deployed as a self-calibration engine. Implemented as a tiny ASIC block, this module learned to correct the pressure sensor's output in real time, counteracting temperature-induced bias. Remarkably, the calibration unit operates on a power budget of only a few nanowatts (about 4.46 nW) and occupies 0.55 mm^2, yet it significantly improves sensor stability over varying thermal conditions. Collectively, these three implementations underscore the central insight of this thesis: by co-designing algorithms and hardware, one can realize ultra-low-power neural accelerators that fit within stringent in-sensor envelopes while still delivering relevant accuracy and responsiveness. This validates in practical scenarios that sensors can indeed host intelligent processing, reducing data transmission needs and latency and enhancing privacy by keeping raw data on-device.

Beyond the application-specific contributions, the thesis includes an *exploratory* study on AI-assisted hardware design. During a research period at Johns Hopkins

University, Large Language Models (LLMs) were evaluated as productivity aids for drafting hardware-description artifacts (e.g., synthesizable RTL designs, testbenches, and documentation) for a recurrent spiking neural network design. This exploratory investigation suggests that LLMs can support early development iterations and reduce boilerplate effort under designer supervision; however, rigorous verification, quantitative assessment of impact, and methodological alignment with constraints-first co-design remain essential prerequisites for adopting such tools in safety- and reliability-critical hardware workflows.

The broader implications of these findings extend to the future of smart sensors and edge AI. The case studies provide a blueprint for embedding machine learning models directly at the sensor level, illustrating that even tasks like audio keyword spotting, mechanical anomaly detection, and sensor self-calibration can be executed locally with negligible energy cost. This localized intelligence can drastically reduce data transmission requirements, enable real-time responsiveness, and safeguard data privacy, all of which are critical as the Internet of Things scales to billions of devices. By proving that custom neural network accelerators can operate in the nanowatt-to-microwatt power regime, this work paves the way for a new class of self-contained, "intelligent" sensor nodes that can perform complex interpretation of raw signals on-site. Such capability transforms sensors from passive data sources to active sentinels that output high-level information instead of unwieldy raw data, easing the burden on downstream networks and cloud analytics. Moreover, the successful adoption of a co-design strategy in multiple domains underscores its generality and transferability – it serves as a practical co-engineering template for others aiming to deploy machine learning in extremely constrained environments. The incorporation of LLM-assisted design also suggests that as neural network models grow more sophisticated, the tools used to implement them can likewise become smarter and more automated, potentially democratizing hardware design for edge AI by lowering the barrier to entry.

Looking forward, the research identifies several open challenges and key directions for advancing tiny neural accelerators and in-sensor intelligence. Robustness is paramount: real-world deployments must cope with variability in operating conditions, manufacturing disparities, and environmental drift. As observed, performance can degrade when a model trained on idealized data encounters out-of-distribution conditions in the field. Future efforts should emphasize robust model training and on-line adaptation, such as calibration policies and lightweight on-device learning schemes that allow sensors to adapt over time without extensive retraining or cloud intervention. Ensuring reliability across the full operational envelope may involve continual self-monitoring of sensor data statistics and triggers for safe model updates (for example, periodic recalibration or federated learning strategies). Another important direction is further miniaturization of the hardware and improvement in energy efficiency. While this thesis's designs are already extremely small, there is room to push the limits down even further, possibly by exploiting mixed-signal circuits or in-memory computing approaches to reduce overheads. Integrating analog pre-processing or non-volatile memory for weight storage could cut power consumption and leakage to near-zero, although such approaches must be co-designed with digital safeguards to handle analog variability. Similarly, exploring advanced semiconductor nodes or 3D integration with sensors could shrink footprints and unlock new levels of integration, albeit with a need to balance technology trade-offs (such as leakage vs.

performance at different nodes).

Reconfigurability and scalability of in-sensor accelerators also present a compelling avenue for research. In future smart sensors, a single hardware accelerator might need to support multiple modes or be repurposed for different tasks as requirements evolve. Designing reconfigurable architectures – for instance, through parameterized datapaths or programmable interconnects – would enable one tiny accelerator to handle a broader class of algorithms or adapt to different sensor modalities. This flexibility must be achieved without sacrificing the efficiency gains of specialization. Additionally, scaling the architectures to higher-bandwidth sensors and multi-sensor systems is an open challenge. The current designs favor extreme resource frugality (often serializing operations to minimize area), but sensors with high output data rates or large sensor arrays (e.g. microphone arrays, multi-axis vibration networks, or distributed environmental sensor suites) will demand more parallel processing and deeper buffering capacity. Research into adaptive acceleration – for example, architectures that can selectively parallelize critical processing when needed, or gracefully power-gate sections of the circuit based on sensor activity – will be key to maintaining energy-proportional operation under varying workloads. Techniques such as multi-rate processing pipelines, event-driven activation (processing data only when significant changes are detected), and hierarchical clock domains for different parts of the circuit can help reconcile the need for higher throughput with the imperative of low energy consumption.

Another frontier highlighted by this work is the improvement of design automation and verification for neural network hardware. Bridging the gap between machine learning models and their hardware implementations remains cumbersome – today it often requires manual intervention to translate quantized neural networks into correct-by-construction RTL, to align numerical behavior (e.g. fixed-point precision, overflow handling) between software and silicon, and to thoroughly verify the hardware against the algorithm. The lack of standardized toolflows or intermediate representations that carry machine-learning-specific metadata (like quantization details or layer scheduling) into hardware design is a bottleneck. Future development of end-to-end toolchains, possibly with open standards, would significantly streamline the path from model to chip. Here, AI-assisted design can play a transformative role: the positive results with ChatGPT suggest that future LLMs or similar AI agents could be integrated more deeply into EDA tools. Beyond generating code, they could automatically propose resource optimizations, generate formal assertions and properties from high-level model descriptions, or analyze synthesis and simulation logs to recommend design refinements. In essence, an AI-in-the-loop design paradigm could emerge, where human designers define constraints and high-level architecture, and AI assistants handle repetitive low-level implementation details and even anticipate corner-case issues. Of course, this raises the importance of thorough verification – when generative models contribute to hardware design, verification and validation frameworks must advance in parallel. Techniques like property checking, constrained-random testing, and equivalence checking between the trained network and the hardware implementation will need to become more automated and integrated, to ensure that the convenience of AI-generated designs does not come at the cost of reliability. Ultimately, combining improved automation with rigorous verification will make the design of tiny neural accelerators more accessible and dependable.

Finally, an important future direction is expanding the scope of smart sensing applications that can benefit from the proposed methodology. The principles of energy-efficient neural co-design demonstrated in this thesis can be extended to a wide array of sensor types and use-cases. For example, upcoming work could involve smart biomedical sensors that preprocess physiological signals (EEG, ECG, etc.) on-device for health monitoring, or autonomous nano-drones with on-board learning for navigation and environmental mapping. Industrial and environmental monitoring systems could integrate multiple sensor modalities (pressure, temperature, gas sensors, cameras) with a common intelligent processing unit, orchestrating a form of multi-modal in-sensor analytics. Applying the lessons learned – such as the importance of data-centric design, careful quantization, and custom-tailored hardware – will be crucial as new domains pose their own unique constraints and reliability requirements. By broadening the application landscape, researchers can also identify any domain-specific challenges not encountered in audio, vibration, or pressure sensing, thereby continually refining the co-design strategy to be even more robust and universal.

In conclusion, this Ph.D. work has charted a path toward tiny, ultra-low-power neural network accelerators that empower sensors with local intelligence. It demonstrates that a careful hardware–software co-design enables even highly resource-constrained devices to perform advanced inference directly at the edge. The synergy of contributions —from concrete hardware prototypes in collaboration with industry, to the proposed constraints-first methodology, and to an exploratory perspective on AI-assisted hardware development—equips the community with both examples and tools to push this vision further. As we look ahead, the convergence of efficient hardware design, machine learning, and emerging AI-assisted development techniques promises to yield a new generation of autonomous, reliable, and adaptable smart sensors. These sensors will not only be smaller and more power-frugal than ever, but also far more capable, ultimately enabling ubiquitous intelligent systems that blend seamlessly into our environments and daily lives. The challenges of robustness, scalability, automation, and integration identified here define a rich research agenda – one that will drive the continued evolution of in-sensor computing from early prototypes into a foundational technology for the future of the Internet of Things. The insights and groundwork laid by this dissertation serve as a stepping stone toward that future, in which sensors are not passive observers but intelligent actors at the frontier of technological innovation.

Bibliography

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2016). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. DOI: 10.48550/arXiv.1603.04467. arXiv: 1603.04467 [cs.DC].

Addabbo, Tommaso, Gian Domenico Licciardo, Riccardo Moretti, Alfredo Rubino, Filippo Spinelli, Valerio Vignoli, and Paola Vitolo (2024). "Monitoring Hardware True Random Number Generators with Artificial Neural Networks: Problem Modeling and Training Dataset Generation". In: *Applications in Electronics Pervading Industry, Environment and Society*. Ed. by Francesco Bellotti, Miltos D. Grammatikakis, Ali Mansour, Massimo Ruo Roch, Ralf Seepold, Agusti Solanas, and Riccardo Berta. Cham: Springer Nature Switzerland, pp. 291–296. ISBN: 978-3-031-48121-5.

Akusok, Anton, Kaj-Mikael Björk, Leonardo Espinosa Leal, Yoan Miche, Renjie Hu, and Amaury Lendasse (2019). "Spiking networks for improved cognitive abilities of edge computing devices". In: *Proceedings of the 12th ACM International Conference on PErvasive Technologies Related to Assistive Environments*. PETRA '19. Rhodes, Greece: Association for Computing Machinery, pp. 307–308. ISBN: 9781450362320. DOI: 10.1145/3316782.3321546. URL: https://doi.org/10.1145/3316782.3321546.

Alajlan, Norah N. and Dina M. Ibrahim (2022). "TinyML: Enabling of Inference Deep Learning Models on Ultra-Low-Power IoT Edge Devices for AI Applications". In: *Micromachines* 13.6. ISSN: 2072-666X. DOI: 10.3390/mi13060851. URL: https://www.mdpi.com/2072-666X/13/6/851.

Ali, Imran, Muhammad Asif, Khuram Shehzad, Muhammad Riaz Ur Rehman, Dong Gyu Kim, Behnam Samadpoor Rikan, YoungGun Pu, Sang Sun Yoo, and Kang-Yoon Lee (2020). "A Highly Accurate, Polynomial-Based Digital Temperature Compensation for Piezoresistive Pressure Sensor in 180 nm CMOS Technology". In: *Sensors* 20.18. ISSN: 1424-8220. DOI: 10.3390/s20185256.

Alwahab, Dhulfiqar A, Dhafer R Zaghar, and Sandor Laki (2018). "FIR Filter Design Based Neural Network". In: *2018 11th International Symposium on Commu-*

nication Systems, Networks Digital Signal Processing (CSNDSP), pp. 1–4. DOI: 10.1109/CSNDSP.2018.8471878.

Alzubaidi, Laith, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan (2021). "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions". In: *Journal of big Data* 8, pp. 1–74.

Ambewadikar, Maheshwari A. and Manasi R. Baheti (2020). "Review on Speech Recognition System for Disabled People Using Automatic Speech Recognition (ASR)". In: *2020 International Conference on Smart Innovations in Design, Environment, Management, Planning and Computing (ICSIDEMPC)*, pp. 31–34. DOI: 10.1109/ICSIDEMPC49020.2020.9299615.

Antolini, Alessio, Francesco Zavalloni, Andrea Lico, Said Quqa, Lorenzo Greco, Mauro Mangia, Fabio Pareschi, Marco Pasotti, and Eleonora Franchi Scarselli (2025). "The Role of Phase-Change Memory in Edge Computing and Analog In-Memory Computing: An Overview of Recent Research Contributions and Future Challenges". In: *Sensors* 25.12. ISSN: 1424-8220. DOI: 10.3390/s25123618. URL: https://www.mdpi.com/1424-8220/25/12/3618.

"Application Specific Integrated Circuit Market Full Report" (2024). In: *market.us*. Accessed: 2024-04-29. URL: https://market.us/report/application-specific-integrated-circuit-market..

Arciniegas, S., D. Rivero, J. Piñan, E. Diaz, and F. Rivas (2025). "IoT device for detecting abnormal vibrations in motors using TinyML". In: *Discover Internet of Things*. DOI: 10.1007/s43926-025-00142-4.

Arm (2021). *Ethos-U55*. https://developer.arm.com/Processors/Ethos-U55. Accessed: 2026-01-12.

Banbury, Colby, Chuteng Zhou, Igor Fedorov, Ramon Matas Navarro, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul N. Whatmough (2021). *MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers*. arXiv: 2010.11267 [cs.LG]. URL: https://arxiv.org/abs/2010.11267.

Belabed, T., M. G. F. Coutinho, M. A. C. Fernandes, V. Carlos, and C. Souani (Sept. 2020). "Low Cost and Low Power Stacked Sparse Autoencoder Hardware Acceleration for Deep Learning Edge Computing Applications". In: *Proc. 5th Int. Conf. on Advanced Technologies for Signal and Image Processing (ATSIP)*, pp. 1–6.

Blocklove, Jason, Siddharth Garg, Ramesh Karri, and Hammond Pearce (2023). "Chip-Chat: Challenges and Opportunities in Conversational Hardware Design". In: *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, pp. 1–6. DOI: 10.1109/MLCAD58807.2023.10299874.

Bolton, Tom, Tooska Dargahi, Sana Belguith, Mabrook S. Al-Rakhami, and Ali Hassan Sodhro (2021). "On the Security and Privacy Challenges of Virtual Assistants". In: *Sensors* 21.7. ISSN: 1424-8220. DOI: 10.3390/s21072312. URL: https://www.mdpi.com/1424-8220/21/7/2312.

Bose, L., P. Dudek, J. Chen, S. J. Carey, and W. Mayol-Cuevas (2020). "Fully embedding fast convolutional networks on pixel processor arrays". In: *European Conference on Computer Vision (ECCV)*, pp. 488–503. DOI: 10.1007/978-3-030-58595-2_30.

Bose, S. K. et al. (Mar. 2020). "ADEPOS: A Novel Approximate Computing Framework for Anomaly Detection Systems and Its Implementation in 65-nm CMOS". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 67.3, pp. 913–926.

Bramanti, Alessia, Angelo Corallo, Gennaro Clemente, Luca Greco, Marina Garofano, Massimo Giordano, Claudio Pascarelli, Gianvito Mitrano, Maria Pia Di Palo, Federica Di Spirito, Massimo Amato, Marianna Bartolomeo, Rosaria Del Sorbo, Michele Ciccarelli, Placido Bramanti, and Pierluigi Ritrovato (2025). "Exploring the Role of Voice Assistants in Managing Noncommunicable Diseases: A Systematic Review on Clinical, Behavioral Outcomes, Quality of Life, and User Experiences". In: *Healthcare* 13.5. ISSN: 2227-9032. DOI: 10.3390/healthcare13050517. URL: https://www.mdpi.com/2227-9032/13/5/517.

Cámbara, Guillermo, Fernando López, David Bonet, Pablo Gómez, Carlos Segura, Mireia Farrús, and Jordi Luque (2022). "TASE: Task-Aware Speech Enhancement for Wake-Up Word Detection in Voice Assistants". In: *Applied Sciences* 12.4. ISSN: 2076-3417. DOI: 10.3390/app12041974. URL: https://www.mdpi.com/2076-3417/12/4/1974.

Capra, M. et al. (2020). "Hardware and Software Optimizations for Accelerating Deep Neural Networks: Survey of Current Trends, Challenges, and the Road Ahead". In: *IEEE Access* 8, pp. 225134–225180.

Case Western Reserve University (CWRU) Bearing Data Center (2020). URL: https://csegroups.case.edu/bearingdatacenter/pages/welcome-case-western-reserve-university-bearing-data-center-website (visited on 08/22/2025).

Cassidy, Andrew S., Julius Georgiou, and Andreas G. Andreou (Sept. 2013). "Design of silicon brains in the nano-CMOS era: Spiking neurons, learning synapses and neural architecture optimization". In: *Neural Networks*. Neuromorphic Engineering: From Neural Systems to Brain-Like Engineered Systems 45, pp. 4–26. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2013.05.011. (Visited on 04/30/2024).

Cavagnis, Leonardo (July 2021). *TinyML: Machine Learning for Embedded System — Part I*. Medium article. URL: https://leonardocavagnis.medium.com/tinyml-machine-learning-for-embedded-system-part-i-92a34529e899.

Chang, Yucheng, Xiaole Cui, Guanting Hou, and Yufeng Jin (2020). "Calibration of the Pressure Sensor Device with the Extreme Learning Machine". In: *2020 21st International Conference on Electronic Packaging Technology (ICEPT)*, pp. 1–5. DOI: 10.1109/ICEPT50128.2020.9202675.

Chaudhuri, Tanaya, Min Wu, Yu Zhang, Pan Liu, and Xiaoli Li (2021). "An Attention-Based Deep Sequential GRU Model for Sensor Drift Compensation". In: *IEEE Sensors Journal* 21.6, pp. 7908–7917. DOI: 10.1109/JSEN.2020.3044388.

Chen, C. C., Z. Liu, G. Yang, C. C. Wu, and Q. Ye (2021). "An Improved Fault Diagnosis Using 1D-Convolutional Neural Network Model". In: *Electronics* 10.1, pp. 1–19.

Chen, Guoguo, Carolina Parada, and Georg Heigold (2014). "Small-footprint keyword spotting using deep neural networks". In: *2014 IEEE International Conference on*

Acoustics, Speech and Signal Processing (ICASSP), pp. 4087–4091. DOI: 10.1109/ICASSP.2014.6854370.

Chen, Yu-Hsin, Tien-Ju Yang, Joel S. Emer, and Vivienne Sze (2017). "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks". In: *IEEE Journal of Solid-State Circuits* 52.1, pp. 127–138. DOI: 10.1109/JSSC.2016.2616357.

Chen, Z. et al. (2020). "Processing near sensor architecture in mixed-signal domain with CMOS image sensor of convolutional-kernel-readout method". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 67.2, pp. 389–400. DOI: 10.1109/TCSI.2019.2949452.

Coelho, Claudionor N., Aki Kuusela, Shan Li, Hao Zhuang, Jennifer Ngadiuba, Thea Klaeboe Aarrestad, Vladimir Loncar, Maurizio Pierini, Adrian Alan Pol, and Sioni Summers (June 2021). "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors". In: *Nature Machine Intelligence* 3.8, pp. 675–686. DOI: 10.1038/s42256-021-00356-5.

Coutinho, M., M. Torquato, and M. Fernandes (2019). "Deep Neural Network Hardware Implementation Based on Stacked Sparse Autoencoder". In: *IEEE Access* 7, pp. 40674–40694.

Cui, M., Y. Wang, X. Lin, and M. Zhong (Feb. 2021). "Fault Diagnosis of Rolling Bearings Based on an Improved Stack Autoencoder and Support Vector Machine". In: *IEEE Sensors Journal* 21.4, pp. 4927–4937.

Da Silva, Bruno, Laurent Segers, An Braeken, Kris Steenhaut, and Abdellah Touhafi (2019). "Design exploration and performance strategies towards power-efficient FPGA-Based architectures for sound source localization". In: *Journal of Sensors*. DOI: 10.1155/2019/5761235.

Datta, Gourav, Souvik Kundu, Zihan Yin, Ravi Teja Lakkireddy, Joe Mathai, Ajey P. Jacob, Peter A. Beerel, and Akhilesh R. Jaiswal (2022). "A processing-in-pixel-in-memory paradigm for resource-constrained TinyML applications". In: *Scientific Reports* 12, p. 14396. DOI: 10.1038/s41598-022-17934-1.

David, Ronald, Jivko Duke, Abhishek Jain, Vijay Janapa Reddi, Nick Jeffries, Jian Li, et al. (2021). *TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems*. arXiv: 2010.08678 [cs.LG]. URL: https://arxiv.org/abs/2010.08678.

De Vita, A., D. Pau, L. Di Benedetto, A. Rubino, F. Petrot, and G. D. Licciardo (Aug. 2020a). "Low Power Tiny Binary Neural Network with Improved Accuracy in Human Recognition Systems". In: *Proc. 23rd Euromicro Conf. on Digital System Design (DSD)*, pp. 309–315.

De Vita, A., D. Pau, C. Parrella, L. Di Benedetto, A. Rubino, and G. D. Licciardo (Aug. 2020b). "Low-Power HW Accelerator for AI Edge-Computing in Human Activity Recognition Systems". In: *Proc. 2nd IEEE Int. Conf. on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 291–295.

De Vita, Antonio, Alessandro Russo, Danilo Pau, Luigi Di Benedetto, Alfredo Rubino, and Gian Domenico Licciardo (2020). "A Partially Binarized Hybrid Neural Network System for Low-Power and Resource Constrained Human Activity Recognition". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 67.11, pp. 3893–3904. DOI: 10.1109/TCSI.2020.3011984.

Deshmukh, Akshay Madhav and Ricardo Chalmeta (2024). "User Experience and Usability of Voice User Interfaces: A Systematic Literature Review". In: *Information* 15.9. ISSN: 2078-2489. DOI: 10.3390/info15090579. URL: https://www.mdpi.com/2078-2489/15/9/579.

Dziadak, B. et al. (2022). "Powering the WSN Node for Monitoring Rail Car Running Gear". In: *Energies*. DOI: 10.3390/en15051641.

Eki, Ryoji, Satoshi Yamada, Hiroyuki Ozawa, Hitoshi Kai, Kazuyuki Okuike, Hareesh Gowtham, Hidetomo Nakanishi, Edan Almog, Yoel Livne, Gadi Yuval, Eli Zyss, and Takashi Izawa (2021). "9.6 A 1/2.3inch 12.3Mpixel with On-Chip 4.97TOPS/W CNN Processor Back-Illuminated Stacked CMOS Image Sensor". In: *2021 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 64, pp. 154–156. DOI: 10.1109/ISSCC42613.2021.9365965.

Eshraghian, Jason K., Max Ward, Emre O. Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D. Lu (2023). "Training Spiking Neural Networks Using Lessons From Deep Learning". In: *Proceedings of the IEEE* 111.9, pp. 1016–1054. DOI: 10.1109/JPROC.2023.3308088.

Fang, W., L. Wang, and P. Ren (2020). "Tinier-YOLO: A Real-Time Object Detection Method for Constrained Environments". In: *IEEE Access* 8, pp. 1935–1944.

Fu, Weimin, Shijie Li, Yifang Zhao, Haocheng Ma, Raj Dutta, Xuan Zhang, Kaichen Yang, Yier Jin, and Xiaolong Guo (2024). "Hardware Phi-1.5B: A Large Language Model Encodes Hardware Domain Specific Knowledge". In: *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 349–354. DOI: 10.1109/ASP-DAC58780.2024.10473927.

Fu, Weimin, Kaichen Yang, Raj Gautam Dutta, Xiaolong Guo, and Gang Qu (2023). "LLM4SecHW: Leveraging Domain-Specific Large Language Model for Hardware Debugging". In: *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 1–6. DOI: 10.1109/AsianHOST59942.2023.10409307.

Futane, N.P., S. Roy Chowdhury, C. Roy Chowdhury, and H. Saha (2010). "ANN based CMOS ASIC design for improved temperature-drift compensation of piezoresistive micro-machined high resolution pressure sensor". In: *Microelectronics Reliability* 50.2, pp. 282–291. ISSN: 0026-2714. DOI: 10.1016/j.microrel.2009.09.012.

Guo, Gangqiang, Bo Chai, Ruichu Cheng, and Yunshuang Wang (2023). "Temperature Drift Compensation of a MEMS Accelerometer Based on DLSTM and ISSA". In: *Sensors* 23.4. ISSN: 1424-8220. DOI: 10.3390/s23041809.

Hamidi, Mohamed, Hassan Satori, Naouar Laaidi, and Khalid Satori (2020). "Conception of Speaker Recognition Methods: A Review". In: *2020 1st International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET)*, pp. 1–6. DOI: 10.1109/IRASET48871.2020.9092118.

Han, Song, Huizi Mao, and William J. Dally (2015a). "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding". In: *arXiv: Computer Vision and Pattern Recognition*.

Han, Song, Jeff Pool, John Tran, and William J. Dally (2015b). "Learning Both Weights and Connections for Efficient Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 28, pp. 1135–1143.

Heydari, Soroush and Qusay H. Mahmoud (2025). "Tiny Machine Learning and On-Device Inference: A Survey of Applications, Challenges, and Future Directions". In: *Sensors* 25.10. ISSN: 1424-8220. DOI: 10.3390/s25103191. URL: https://www.mdpi.com/1424-8220/25/10/3191.

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean (2015). *Distilling the Knowledge in a Neural Network*. arXiv: 1503.02531 [cs.CL]. URL: https://arxiv.org/abs/1503.02531.

Hogenauer, E. (1981). "An economical class of digital filters for decimation and interpolation". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 29.2, pp. 155–162. DOI: 10.1109/TASSP.1981.1163535.

Howard, Andrew G., Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. arXiv: 1704.04861 [cs.CV]. URL: https://arxiv.org/abs/1704.04861.

Hsu, Tzu-Hsiang, Guan-Cheng Chen, Yi-Ren Chen, Chung-Chuan Lo, Ren-Shuo Liu, Meng-Fan Chang, Kea-Tiong Tang, and Chih-Cheng Hsieh (2022). "A 0.8V Intelligent Vision Sensor with Tiny Convolutional Neural Network and Programmable Weights Using Mixed-Mode Processing-in-Sensor Technique for Image Classification". In: *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 65, pp. 1–3. DOI: 10.1109/ISSCC42614.2022.9731675.

Huang, Guyue, Jingbo Hu, Yifan He, Jialong Liu, Mingyuan Ma, Zhaoyang Shen, Juejian Wu, Yuanfan Xu, Hengrui Zhang, Kai Zhong, Xuefei Ning, Yuzhe Ma, Haoyu Yang, Bei Yu, Huazhong Yang, and Yu Wang (June 2021). "Machine Learning for Electronic Design Automation: A Survey". In: *ACM Trans. Des. Autom. Electron. Syst.* 26.5. ISSN: 1084-4309. DOI: 10.1145/3451179. URL: https://doi.org/10.1145/3451179.

Iandola, Forrest N., Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer (2016). *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size*. arXiv: 1602.07360 [cs.CV]. URL: https://arxiv.org/abs/1602.07360.

IPC/JEDEC (2004). *Moisture/Reflow Sensitivity Classification for Non-Hermetic Solid State Surface Mount Devices*. Tech. rep. J-STD-020C. IPC and JEDEC Solid State Technology Association.

Jacob, Benoit, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko (2018). "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713. DOI: 10.1109/CVPR.2018.00286.

Jamuna, G., S. Yellampalli, and S. Swetha (2014). "Design and Implementation of Telescopic OTA in 8-Bit Second-Order Continuous-Time Band-Pass Sigma–Delta ADC". In: *Proceedings of the International Conference on Electrical, Communication and Computer Engineering*. Proc. Int. Conf. Electr. Commun. Comput. Eng. Hosur, India, pp. 1–7.

Kar, B., P. K. Gopalakrishnan, S. K. Bose, M. Roy, and A. Basu (Dec. 2020). "ADIC: Anomaly Detection Integrated Circuit in 65-nm CMOS Utilizing Approximate Computing". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.12, pp. 2518–2529.

Koh, Min-Sung (2021). "Learnable Linear Phase FIR Filter Designs Using a Generative Adversarial Network". In: *2021 15th International Conference on Signal Processing and Communication Systems (ICSPCS)*, pp. 1–8. DOI: 10.1109/ICSPCS53099.2021.9660300.

Kolok, P. et al. (2025). "Low-Cost IoT-Based Predictive Maintenance Using Vibration". In: *Sensors*. DOI: 10.3390/s25216610.

Kouritzin, Michael A. and Daniel Richard (2024). "Universal Approximation and the Topological Neural Network". In: *IEEE Access* 12, pp. 115064–115084. DOI: 10.1109/ACCESS.2023.3342063.

Lai, Liangzhen, Naveen Suda, and Vikas Chandra (2018). *CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs*. arXiv: 1801.06601 [cs.AR]. URL: https://arxiv.org/abs/1801.06601.

LARQ Documentation (2021). URL: https://docs.larq.dev/larq/ (visited on 08/22/2025).

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: 10.1109/5.726791.

Lei, Y., Q. Deng, S. Long, S. Liu, and S. Oh (Dec. 2020). "An Effective Design to Improve the Efficiency of DPUs on FPGA". In: *Proc. IEEE 26th Int. Conf. on Parallel and Distributed Systems (ICPADS)*, pp. 206–213.

Lewis, Jerad (2013). *Technical article: Analog and Digital MEMS Microphone Design Considerations*. Technical article MS-2472. TA11709-0-8/13. 4 pp. URL: https://www.analog.com/media/en/technical-documentation/technical-articles/analog-and-digital-mems-microphone-design-considerations-ms-2472.pdf.

Li, Fanyang, Tao Yin, Shuwen Wu, and Wenren Deng (2023). "A Compact MEMS Microphone Digital Readout System Using LDO and PPA-Less VCO-Based Delta-Sigma Modulation Technique". In: *Electronics* 12.24. ISSN: 2079-9292. DOI: 10.3390/electronics12245014. URL: https://www.mdpi.com/2079-9292/12/24/5014.

Li, Yunyao, Jing He, Zhijun Xie, Dacheng Jiang, and Youtao Li (Jan. 2018). "Frontier development of chips design and production". In: *Procedia Computer Science* 139, pp. 554–560. DOI: 10.1016/j.procs.2018.10.232.

Li, Zewen, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou (2022). "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.12, pp. 6999–7019. DOI: 10.1109/TNNLS.2021.3084827.

Licciardo, Gian Domenico, Paola Vitolo, Stefano Bosco, Santo Pennino, Danilo Pau, Massimo Pesaturo, Luigi Di Benedetto, and Rosalba Liguori (2023). "Ultra-Tiny Neural Network for Compensation of Post-soldering Thermal Drift in MEMS Pressure Sensors". In: *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5. DOI: 10.1109/ISCAS46773.2023.10181480.

LiKamWa, R., Y. Hou, J. Gao, M. Polansky, and L. Zhong (2016). "RedEye: Analog ConvNet image sensor architecture for continuous mobile vision". In: *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, pp. 255–266. DOI: 10.1109/ISCA.2016.28.

Lin, Ji, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han (2020a). "MCUNet: Tiny Deep Learning on IoT Devices". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33.

Lin, Ji, Wei-Ming Chen, Yujun Lin, john cohn john, Chuang Gan, and Song Han (2020b). "MCUNet: Tiny Deep Learning on IoT Devices". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., pp. 11711–11722. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/86c51678350f656dcc7f490a43946ee5-Paper.pdf.

Lin, Ji, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, and Song Han (2023). "Tiny Machine Learning: Progress and Futures [Feature]". In: *IEEE Circuits and Systems Magazine* 23.3, pp. 8–34. DOI: 10.1109/MCAS.2023.3302182.

Liu, H., D. Yao, J. Yang, and X. Li (2019). "Lightweight Convolutional Neural Network and Its Application in Rolling Bearing Fault Diagnosis Under Variable Working Conditions". In: *Sensors* 19.22, pp. 1–20.

Liu, Mingjie, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren (2023). "Invited Paper: VerilogEval: Evaluating Large Language Models for Verilog Code Generation". In: *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–8. DOI: 10.1109/ICCAD57390.2023.10323812.

Losada, M., A. Olaizola, A. Irizar, et al. (2024). "UWB-Based Accelerometer Sensor Nodes for Low-Power Applications in Offshore Platforms". In: *Electronics*. DOI: 10.3390/electronics13224485.

Ma, Z. et al. (2019). "An analog near-sensor computing architecture featuring a current-mode low-precision binary neural network". In: *IEEE Asian Solid-State Circuits Conference (A-SSCC)*, pp. 65–66. DOI: 10.1109/A-SSCC47793.2019.9055872.

Magar, R., L. Ghule, J. Li, Y. Zhao, and A. B. Farimani (2021). "FaultNet: A Deep Convolutional Neural Network for Bearing Fault Classification". In: *IEEE Access* 9, pp. 25189–25199.

Maria, J., J. Amaro, G. Falcão, and L. A. Alexandre (Apr. 2016). "Stacked Autoencoders Using Low-Power Accelerated Architectures for Object Recognition in Autonomous Systems". In: *Neural Processing Letters* 43.2, pp. 445–458.

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Martínez Lahoz, Javier, David Asiain Ansorena, and José Ramón Beltrán Blázquez (2023). "Impact of Thermal Variations and Soldering Process on Performance and Behavior of MEMS Capacitive Accelerometers". In: *IEEE Sensors Journal* 23.22, pp. 27124–27136. DOI: 10.1109/JSEN.2023.3321238.

Meech, James Timothy (2023). "Leveraging High-Level Synthesis and Large Language Models to Generate, Simulate, and Deploy a Uniform Random Number Generator Hardware Design". In: *arXiv:2311.03489* abs/2311.03489. URL: https://api.semanticscholar.org/CorpusID:265043065.

Mennel, L. et al. (2020). "Ultrafast machine vision with 2D material neural network image sensors". In: *Nature* 579, pp. 62–66. DOI: 10.1038/s41586-020-2047-9.

MLCommons (2021). *Pre-trained Audio Wakeword Models*. Version v0.5, GitHub repository. URL: https://github.com/mlcommons/tiny/tree/v0.5/v0.5/training/keyword_spotting/trained_models (visited on 08/22/2025).

Mohammed, Sedir, Lukas Budach, Moritz Feuerpfeil, Nina Ihde, Andrea Nathansen, Nele Noack, Hendrik Patzlaff, Felix Naumann, and Hazar Harmouch (2025). "The effects of data quality on machine learning performance on tabular data". In: *Information Systems* 132, p. 102549. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2025.102549. URL: https://www.sciencedirect.com/science/article/pii/S0306437925000341.

Molchanov, Pavlo, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz (2017). *Pruning Convolutional Neural Networks for Resource Efficient Inference*. arXiv: 1611.06440 [cs.CV]. URL: https://arxiv.org/abs/1611.06440.

Murphy, C. (2025). *Choosing the Most Suitable Predictive Maintenance Sensor*. Analog Devices. URL: https://www.analog.com/en/technical-articles/choosing-the-most-suitable-predictive-maintenance-sensor.html (visited on 08/22/2025).

Murray, Thomas S., Daniel R. Mendat, Kayode A. Sanni, Philippe O. Pouliquen, and Andreas G. Andreou (2018). "Bio-Inspired Human Action Recognition With a Micro-Doppler Sonar System". In: *IEEE Access* 6, pp. 28388–28403. DOI: 10.1109/ACCESS.2017.2732919.

Musa, Aminu, Habeebah Adamu Kakudi, Mohammed Hassan, Mohamed Hamada, Usman Umar, and Maryam Lawan Salisu (2025). "Lightweight Deep Learning Models For Edge Devices—A Survey". In: *International Journal of Computer Information Systems and Industrial Management Applications* 17, p. 18. DOI: 10.70917/ijcisim-2025-0014.

Najar, Hadi (2019). "Electrical Only Calibration of Barometric Pressure Sensors Using Machine Learning". In: *2019 20th International Conference on Solid-State Sensors, Actuators and Microsystems & Eurosensors XXXIII (TRANSDUCERS & EUROSENSORS XXXIII)*, pp. 1977–1980. DOI: 10.1109/TRANSDUCERS.2019.8808801.

NASA Prognostics Data Repository (2025). NASA Ames Research Center. URL: https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/ (visited on 08/22/2025).

Ouyang, Bangsen, Jialiang Wang, Guang Zeng, Jianmin Yan, Yue Zhou, Xixi Jiang, Bangjie Shao, and Yang Chai (2024). "Bioinspired in-sensor spectral adaptation for perceiving spectrally distinctive features". In: *Nature Electronics* 7.8, pp. 705–713. DOI: 10.1038/s41928-024-01208-x. URL: https://doi.org/10.1038/s41928-024-01208-x.

Pan, Wen, Jiyuan Zheng, Lai Wang, and Yi Luo (2022). "A Future Perspective on In-Sensor Computing". In: *Engineering*. ISSN: 2095-8099. DOI: https://doi.org/10.1016/j.eng.2022.01.009.

Papavasileiou, Apostolis, Stelios Nikoladakis, Fotios Panagiotis Basamakis, Sotiris Aivaliotis, George Michalos, and Sotiris Makris (2024). "A Voice-Enabled ROS2 Framework for Human–Robot Collaborative Inspection". In: *Applied Sciences* 14.10. ISSN: 2076-3417. DOI: 10.3390/app14104138. URL: https://www.mdpi.com/2076-3417/14/10/4138.

Pappalardo, Alessandro et al. (Feb. 2024). *Xilinx/brevitas: v0.10.2*. Version v0.10.2. DOI: 10.5281/zenodo.10680356. URL: https://doi.org/10.5281/zenodo.10680356.

Patra, J.C., V. Gopalkrishnan, Ee Luang Ang, and A. Das (2004). "Neural network-based self-calibration/compensation of sensors operating in harsh environments [smart pressure sensor example]". In: *SENSORS, 2004 IEEE*, 425–428 vol.1. DOI: 10.1109/ICSENS.2004.1426190.

Pau, Danilo, Welid Ben Yahmed, Fabrizio Maria Aymone, Gian Domenico Licciardo, and Paola Vitolo (2023). "Tiny Machine Learning Zoo for Long-Term Compensation of Pressure Sensor Drifts". In: *Electronics* 12, 4819.23. ISSN: 2079-9292. DOI: 10.3390/electronics12234819.

Pau, Danilo Pietro, Welid Ben Yahmed, Stefano Bosco, Gian Domenico Licciardo, Santo Pennino, and Paola Vitolo (2024a). "Ultra-Tiny Quantized Neural Networks for Piezo Pressure Sensors". In: *2024 Smart Systems Integration Conference and Exhibition (SSI)*, pp. 1–8. DOI: 10.1109/SSI63222.2024.10740508.

Pau, Danilo Pietro, Welid Ben Yahmed, Stefano Bosco, Santo Pennino, Gian Domenico Licciardo, and Paola Vitolo (2024b). "Ultra Tiny Neural Network for Accurate Pressure Sensors Under Multiple Thermal Stresses". In: *2024 IEEE 8th Forum on Research and Technologies for Society and Industry Innovation (RTSI)*, pp. 607–612. DOI: 10.1109/RTSI61910.2024.10761275.

Pinkham, Reid, Andrew Berkovich, and Zhengya Zhang (2021). "Near-Sensor Distributed DNN Processing for Augmented and Virtual Reality". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11.4, pp. 663–676. DOI: 10.1109/JETCAS.2021.3121259.

Ran, Y., X. Zhou, P. Lin, Y. Wen, and R. Deng (2019). "A Survey of Predictive Maintenance: Systems, Purposes and Approaches". In: arXiv: 1912.07383 [cs.LG].

Ray, P. P. (2022). "A review on TinyML: State-of-the-art and prospects". In: *Journal of King Saud University - Computer and Information Sciences* 34.6, pp. 1595–1623. DOI: 10.1016/j.jksuci.2020.10.014.

Reddi, Vijay Janapa, Colby Banbury, Peter Torelli, Joshua Holleman, Nick Jeffries, Csaba Király, et al. (2021). *MLPerf Tiny Benchmark*. arXiv: 2106.07597 [cs.LG]. URL: https://arxiv.org/abs/2106.07597.

Ribeiro, M., M. Gutoski, A. E. Lazzaretti, and H. S. Lopes (2020). "One-Class Classification in Images and Videos Using a Convolutional Autoencoder with Compact Embedding". In: *IEEE Access* 8, pp. 86520–86535.

Rivera, José, Mariano Carrillo, Mario Chacón, Gilberto Herrera, and Gilberto Bojorquez (2007). "Self-calibration and optimal response in intelligent sensors design based on artificial neural networks". In: *Sensors* 7.8, pp. 1509–1529. DOI: 10.3390/s7081509.

Rogdakis, Konstantinos, George Psaltakis, Giorgos Fagas, Aidan Quinn, Rodrigo Martins, and Emmanuel Kymakis (Feb. 21, 2024). "Hybrid chips to enable a sustainable internet of things technology: opportunities and challenges". In: *Discover Materials* 4.1, p. 4. ISSN: 2730-7727. DOI: 10.1007/s43939-024-00074-w. URL: https://doi.org/10.1007/s43939-024-00074-w.

Rosário, Albérico Travassos and Joana Carmo Dias (Jan. 2023). "How Industry 4.0 and Sensors Can Leverage Product Design: Opportunities and Challenges". In: *Sensors* 23.3, p. 1165. ISSN: 1424-8220. DOI: 10.3390/s23031165.

Samanta, Riya, Bidyut Saha, Soumya K. Ghosh, and Ram Babu Roy (2024). *Optimizing TinyML: The Impact of Reduced Data Acquisition Rates for Time Series Classification on Microcontrollers*. arXiv: 2409.10942 [cs.LG]. URL: https://arxiv.org/abs/2409.10942.

Sandler, Mark, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen (2018). "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

Sapsanis, Christos, Nathaniel Welsh, Michael Pozin, Guillaume Garreau, Gaspar Tognetti, Hani Bakhshaee, Philippe O. Pouliquen, Rajat Mitral, William R. Thompson, and Andreas G. Andreou (2018). "StethoVest: A simultaneous multichannel wearable system for cardiac acoustic mapping". In: *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1–4. DOI: 10.1109/BIOCAS.2018.8584742.

Sari, E., M. Belbahri, and V. Partovi Nia (2019). "How Does Batch Normalization Help Binary Training?" In: arXiv: 1909.09139 [cs.LG].

Saufi, S. R., Z. Ahmad, M. S. Leong, and M. H. Lim (2019). "Challenges and Opportunities of Deep Learning Models for Machinery Fault Detection and Diagnosis: A Review". In: *IEEE Access* 7, pp. 122644–122662.

Schreier, R. (2020). "Delta Sigma Toolbox". In: *MATLAB Central File Exchange*. URL: https://www.mathworks.com/matlabcentral/fileexchange/19-delta-sigma-toolbox.

Sindhu, Irum and Mohd Shamrie Sainin (2024). "Automatic Speech and Voice Disorder Detection Using Deep Learning—A Systematic Literature Review". In: *IEEE Access* 12, pp. 49667–49681. DOI: 10.1109/ACCESS.2024.3371713.

Solera-Cotanilla, Sonia, Mario Vega-Barbas, Jaime Pérez, Gregorio López, Javier Matanza, and Manuel Álvarez-Campana (2022). "Security and Privacy Analysis of Youth-Oriented Connected Devices". In: *Sensors* 22.11. ISSN: 1424-8220. DOI: 10.3390/s22113967. URL: https://www.mdpi.com/1424-8220/22/11/3967.

Song, Ruibing, Kejie Huang, Zongsheng Wang, and Haibin Shen (2022). "A Reconfigurable Convolution-in-Pixel CMOS Image Sensor Architecture". In: *IEEE Transactions on Circuits and Systems for Video Technology* 32.10, pp. 7212–7225. DOI: 10.1109/TCSVT.2022.3179370.

Spinelli, F., R. Moretti, T. Addabbo, P. Vitolo, and G. D. Licciardo (2023). "Low-complexity Machine Learning Architecture for Hardware-aware True Random Number Generators Assessment and Continuous Monitoring". In: *2023 18th Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*, pp. 221–224. DOI: 10.1109/PRIME58259.2023.10161903.

STMicroelectronics (2015). *Ultra-Low-Power High-Performance 3-Axis Accelerometer with Digital Output for Industrial Applications*. DocID027668 Rev 2. STMicroelectronics.

STMicroelectronics (2017). *Application note: Tutorial for MEMS microphones*. Application note AN4426. DocID025704 Rev. 2. STMicroelectronics. 20 pp.

STMicroelectronics (2019). "High-performance MEMS nano pressure sensor: 260-1260 hPa absolute digital output barometer". In: *LPS22HH datasheet*.

STMicroelectronics (2020). *Ultrasound Behavior and Guidelines of Analog MEMS Microphone IMP23ABSU*. IMP23ABSU Rev 2. STMicroelectronics.

STMicroelectronics (2021). "MEMS audio sensor omnidirectional digital microphone for industrial applications. IMP34DT05-Rev 4". In.

STMicroelectronics (2024a). *Introducing ST Neural-ART Accelerator (Product Presentation)*. https://www.st.com/resource/en/product_presentation/st-neural-art-accelerator-introduction.pdf. Accessed: 2026-01-12.

STMicroelectronics (2024b). *STM32N6 Series*. https://www.st.com/en/microcontrollers-microprocessors/stm32n6-series.html. Accessed: 2026-01-12.

Stošić, Biljana P. (2021). "Improved Classes of CIC Filter Functions: Design and Analysis of the Quantized-Coefficient Errors". In: *2021 56th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST)*, pp. 65–68. DOI: 10.1109/ICEST52640.2021.9483471.

Sze, Vivienne, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer (2017). "Efficient Processing of Deep Neural Networks: A Tutorial and Survey". In: *Proceedings of the IEEE* 105.12, pp. 2295–2329.

Tan, Mingxing, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, and Quoc V. Le (2019). "MnasNet: Platform-Aware Neural Architecture Search for Mobile". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2820–2828.

Tang, X. et al. (2023). "A miniature and intelligent low-power in situ wireless monitoring system for wheel alignment". In: *Measurement*. DOI: 10.1016/j.measurement.2023.112578.

Thakur, Shailja, Baleegh Ahmad, Zhenxing Fan, Hammond Pearce, Benjamin Tan, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg (2023). "Benchmarking Large Language Models for Automated Verilog RTL Code Generation". In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6. DOI: 10.23919/DATE56975.2023.10137086.

Thakur, Shailja, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg (Apr. 2024). "VeriGen: A Large Language Model for Verilog Code Generation". In: *ACM Trans. Des. Autom. Electron. Syst.* 29.3. ISSN: 1084-4309. DOI: 10.1145/3643681. URL: https://doi.org/10.1145/3643681.

Tomlinson, Michael, Joe Li, and Andreas Andreou (2024). "Designing Silicon Brains Using LLM: Leveraging ChatGPT for Automated Description of a Spiking Neuron Array". In: *2024 Argentine Conference on Electronics (CAE)*, pp. 154–159. DOI: 10.1109/CAE59785.2024.10487167.

Tran, Anh Vang, Xianmin Zhang, and Benliang Zhu (2019). "Effects of Temperature and Residual Stresses on the Output Characteristics of a Piezoresistive Pressure Sensor". In: *IEEE Access* 7, pp. 27668–27676. DOI: 10.1109/ACCESS.2019.2901846.

Tsukada, M., M. Kondo, and H. Matsutani (July 2020). "A Neural Network-Based On-Device Learning Anomaly Detector for Edge Devices". In: *IEEE Transactions on Computers* 69.7, pp. 1027–1044.

Varshney, A., N. Garg, K. S. Nagla, T. S. Nair, S. K. Jaiswal, S. Yadav, and D. K. Aswal (June 2021). "Challenges in Sensors Technology for Industry 4.0 for Futuristic Metrological Applications". In: *MAPAN* 36.2, pp. 215–226. ISSN: 0974-9853. DOI: 10.1007/s12647-021-00453-1.

Vincze, Miklos, Bela Molnar, and Miklos Kozlovszky (2025). "The Use of Voice Control in 3D Medical Data Visualization Implementation, Legal, and Ethical Issues". In: *Information* 16.1. ISSN: 2078-2489. DOI: 10.3390/info16010012. URL: https://www.mdpi.com/2078-2489/16/1/12.

Vitolo, Paola, Antonio De Vita, Luigi Di Benedetto, Danilo Pau, and Gian Domenico Licciardo (2022a). "Low-Power Detection and Classification for In-Sensor Predictive Maintenance Based on Vibration Monitoring". In: *IEEE Sensors Journal* 22.7, pp. 6942–6951. DOI: 10.1109/JSEN.2022.3154479.

Vitolo, Paola, Pio Esposito, Danilo Pau, Rosalba Liguori, Luigi Di Benedetto, and Gian Domenico Licciardo (2023a). "In-sensor neural network for real-time KWS by image processing". In: *Real-time Processing of Image, Depth and Video Information 2023*. Vol. 12571. International Society for Optics and Photonics. SPIE. DOI: 10.1117/12.2665545.

Vitolo, Paola, Gian Domenico Licciardo, Anna Chiara Amendola, Luigi Di Benedetto, Rosalba Liguori, Alfredo Rubino, and Danilo Pau (2022b). "Quantized ID-CNN for a Low-power PDM-to-PCM Conversion in TinyML KWS Applications". In: *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 154–157. DOI: 10.1109/AICAS54282.2022.9869909.

Vitolo, Paola, Gian Domenico Licciardo, Luigi di Benedetto, Rosalba Liguori, Alfredo Rubino, and Danilo Pau (2021). "Low-Power Anomaly Detection and Classification System based on a Partially Binarized Autoencoder for In-Sensor Computing". In: *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pp. 1–5. DOI: 10.1109/ICECS53924.2021.9665479.

Vitolo, Paola, Gian Domenico Licciardo, Rosalba Liguori, Luigi Di Benedetto, Alfredo Rubino, Danilo Pau, and Massimo Pesaturo (2024a). "In-Sensor System for Real-Time Compensation of Thermal Drift in MEMS Pressure Sensors". In: *Proceedings of SIE 2023*. Ed. by Carmine Ciofi and Ernesto Limiti. Cham: Springer Nature Switzerland, pp. 186–191. ISBN: 978-3-031-48711-8.

Vitolo, Paola, Gian Domenico Licciardo, Danilo Pau, Rosalba Liguori, Luigi Di Benedetto, and Alfredo Rubino (2024b). "In-Sensor Self-Calibration Circuit of MEMS Pressure Sensors for Accurate Localization". In: *2024 27th Euromicro Conference on Digital System Design (DSD)*, pp. 582–587. DOI: 10.1109/DSD64264.2024.00083.

Vitolo, Paola, Rosalba Liguori, Luigi Di Benedetto, Alfredo Rubino, and Gian Domenico Licciardo (2024c). "Automatic Audio Feature Extraction for Keyword Spotting". In: *IEEE Signal Processing Letters* 31, pp. 161–165. DOI: 10.1109/LSP.2023.3346280.

Vitolo, Paola, Rosalba Liguori, Luigi Di Benedetto, Alfredo Rubino, Danilo Pau, and Gian Domenico Licciardo (2023b). "A 0.8 mW TinyML-Based PDM-to-PCM Conversion for In-Sensor KWS Applications". In: *Proceedings of SIE 2022*. Ed. by Giuseppe Cocorullo, Felice Crupi, and Ernesto Limiti. Cham: Springer Nature Switzerland, pp. 146–151. ISBN: 978-3-031-26066-7. DOI: 10.1007/978-3-031-26066-7_23.

Vitolo, Paola, Rosalba Liguori, Luigi Di Benedetto, Alfredo Rubino, Danilo Pau, and Gian Domenico Licciardo (2023c). "A New NN-Based Approach to In-Sensor PDM-to-PCM Conversion for Ultra TinyML KWS". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 70.4, pp. 1595–1599. DOI: 10.1109/TCSII.2022.3224022.

Vitolo, Paola, Rosalba Liguori, Luigi Di Benedetto, Alfredo Rubino, Danilo Pau, and Gian Domenico Licciardo (2025). "Real-time neural network-based thermal stress compensation for pressure sensors in precision localization systems". In: *Microprocessors and Microsystems* 117, p. 105183. ISSN: 0141-9331. DOI: https://doi.org/10.1016/j.micpro.2025.105183.

Vitolo, Paola, Danilo Pau, Gian Domenico Licciardo, Massimiliano Pesaturo, Stefano Bosco, and Santo Pennino (2023d). "Tiny compensation of pressure drift measurements due to long exposures to high temperatures". In: *2023 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pp. 01–05. DOI: 10.1109/I2MTC53148.2023.10175998.

Vitolo, Paola, George Psaltakis, Michael Tomlinson, Gian Domenico Licciardo, and Andreas G. Andreou (2024d). "Natural Language to Verilog: Design of a Recurrent Spiking Neural Network using Large Language Models and ChatGPT". In: *2024 International Conference on Neuromorphic Systems (ICONS)*, pp. 110–116. DOI: 10.1109/ICONS62911.2024.00024.

Vitolo, Paola, Michael Tomlinson, ChatGPT-4, Gian Domenico, and Andreas Andreou (2024e). "Tiny Tapeout 06: ChatGPT designed Recurrent Spiking Neural Network". In: *Andreou JHU Lab*. Accessed: 2024-04-29. URL: https://github.com/Andreou-JHULabOrg/tinytapeout%5C_06%5C_chatgpt%5C_rsnn.

Wan, Lily Jiaxin, Yingbing Huang, Yuhong Li, Hanchen Ye, Jinghua Wang, Xiaofan Zhang, and Deming Chen (2024). "Invited Paper: Software/Hardware Co-design for LLM and Its Application for Design Verification". In: *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 435–441. DOI: 10.1109/ASP-DAC58780.2024.10473893.

Wan, Renzhuo, Yuandong Li, Chengde Tian, Fan Yang, Wendi Deng, Siyu Tang, Jun Wang, and Wei Zhang (2022). "Design and Implementation of Sigma-Delta ADC Filter". In: *Electronics* 11.24. ISSN: 2079-9292. DOI: 10.3390/electronics11244229. URL: https://www.mdpi.com/2079-9292/11/24/4229.

Wang, L. et al. (2025). "Self-powered wireless sensor node with hybrid energy harvesting for railway bearing fault diagnosis". In: *Information Fusion*. DOI: 10.1016/j.inffus.2025.102630.

Wang, Tao, Jiawen Ji, Jianglong Lan, and Bo Wang (2025). "Ultrasonic Beamforming-Based Visual Localisation of Minor and Multiple Gas Leaks Using a Microelectromechanical System (MEMS) Microphone Array". In: *Sensors* 25.10. ISSN: 1424-8220. DOI: 10.3390/s25103190. URL: https://www.mdpi.com/1424-8220/25/10/3190.

Wang, Tao, Pengyu Liu, Wenjing Zhang, Xiaowei Jia, Yanming Wang, and Jiachun Yang (2022). "Calibration of Multi-dimensional Air Pressure Sensor Based on LSTM". In: *Artificial Intelligence and Security*. Ed. by Xingming Sun, Xiaorui Zhang, Zhihua Xia, and Elisa Bertino. Cham: Springer International Publishing,

pp. 532–543. ISBN: 978-3-031-06791-4. DOI: 10.1007/978-3-031-06791-4_42.

Wang, Yannan, Baoxin Xu, Marcin Chrzanowski, David Blaauw, and Hun-Seok Kim (2018). "Energy Efficient Binary Neural Network with Reordered Operations and Custom Bit Precision". In: *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*.

Wang, Yizhi, Jun Lin, and Zhongfeng Wang (2018). "An Energy-Efficient Architecture for Binary Weight Convolutional Neural Networks". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.2, pp. 280–293. DOI: 10.1109/TVLSI.2017.2767624.

Wang, Z., Q. Zhang, J. Xiong, M. Xiao, G. Sun, and J. He (Sept. 2017). "Fault Diagnosis of a Rolling Bearing Using Wavelet Packet Denoising and Random Forests". In: *IEEE Sensors Journal* 17.17, pp. 5581–5588.

Wang, Zihan, Junwei Xie, Zhiping Yao, Xu Kuang, Qinquan Gao, and Tong Tong (2022). "NFFKD: A Knowledge Distillation Method Based on Normalized Feature Fusion Model". In: *2022 IEEE 5th International Conference on Big Data and Artificial Intelligence (BDAI)*, pp. 111–116. DOI: 10.1109/BDAI56143.2022.9862657.

Warden, Pete (2018). *Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition*. DOI: 10.48550/ARXIV.1804.03209. URL: https://arxiv.org/abs/1804.03209.

Whang, Steven Euijong, Yuji Roh, Hwanjun Song, and Jae-Gil Lee (2023). "Data Collection and Quality Challenges in Deep Learning: A Data-Centric AI Perspective". In: *The VLDB Journal* 32.4, pp. 791–813. DOI: 10.1007/s00778-022-00775-9.

Woo, Jongseok, Kuchul Jung, and Saibal Mukhopadhyay (2024). "Efficient Hardware Design of DNN for RF Signal Modulation Recognition Employing Ternary Weights". In: *IEEE Access* 12, pp. 80165–80175. DOI: 10.1109/ACCESS.2024.3409180.

Woo, Sangwon, Gihun Lee, Jaeho Lee, and Hoi-Jun Yoo (2024). "TernaryBERT: Reducing Weight Precision to Ternary in BERT-like Models for Efficient Inference". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 71.1.

Wu, Bichen, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer (2019). "FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable NAS". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10734–10742.

Wu, Ruirui, Huaijiang Li, and Lei Gao (Mar. 2023). "Research on temperature drift mechanism and compensation method of silicon piezoresistive pressure sensors". In: *AIP Advances* 13.3, p. 035323. ISSN: 2158-3226. DOI: 10.1063/5.0135401.

Xie, Jinlu, Zhitian Li, and Xudong Zou (2023). "Dynamic Temperature Compensation of Pressure Sensors in Migratory Bird Biologging Applications". In: *Electronics* 12.20. ISSN: 2079-9292. DOI: 10.3390/electronics12204373.

Xilinx (Feb. 2018). *Data Sheet of the 7 Series FPGAs: Overview (XC7A35T-1CPG236C)*. URL: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf (visited on 08/22/2025).

Yamanaka, Y., T. Iwata, H. Takahashi, M. Yamada, and S. Kanai (2019). "Autoencoding Binary Classifiers for Supervised Anomaly Detection". In: *Trends in Artificial Intelligence*. Vol. 11671. Lecture Notes in Computer Science. Springer, pp. 647–659.

Yang, J., Y. Yang, and G. Xie (Aug. 2020). "Diagnosis of Incipient Fault Based on Sliding-Scale Resampling Strategy and Improved Deep Autoencoder". In: *IEEE Sensors Journal* 20.15, pp. 8336–8348.

Yu, W. and P. Lv (2021). "An End-to-End Intelligent Fault Diagnosis Application for Rolling Bearing Based on MobileNet". In: *IEEE Access* 9, pp. 41925–41933.

Zaman, Khalid, Melike Sah, Cem Direkoglu, and Masashi Unoki (2023). "A Survey of Audio Classification Using Deep Learning". In: *IEEE Access* 11, pp. 106620–106649. DOI: 10.1109/ACCESS.2023.3318015.

Zhai, J., S. Zhang, J. Chen, and Q. He (Oct. 2018). "Autoencoder and Its Various Variants". In: *Proc. IEEE Int. Conf. Systems, Man, and Cybernetics (SMC)*. Miyazaki, Japan, pp. 415–419.

Zhang, S., F. Ye, B. Wang, and T. G. Habetler (Mar. 2021). "Semi-Supervised Bearing Fault Diagnosis and Classification Using Variational Autoencoder-Based Deep Generative Models". In: *IEEE Sensors Journal* 21.5, pp. 6476–6486.

Zhang, S., S. Zhang, B. Wang, and T. G. Habetler (2020). "Deep Learning Algorithms for Bearing Fault Diagnostics—A Comprehensive Review". In: *IEEE Access* 8, pp. 29857–29881.

Zheng, Zhuoyue, Chen Wang, Linlin Wang, Zeyu Ji, Xiaoxiao Song, Pui-In Mak, Huafeng Liu, and Yuan Wang (2024). "Micro-Electro-Mechanical Systems Microphones: A Brief Review Emphasizing Recent Advances in Audible Spectrum Applications". In: *Micromachines* 15.3. ISSN: 2072-666X. DOI: 10.3390/mi15030352. URL: https://www.mdpi.com/2072-666X/15/3/352.

Zhou, Feichi and Yang Chai (Nov. 2020). "Near-sensor and in-sensor computing". In: *Nature Electronics* 3, pp. 664–671. DOI: 10.1038/s41928-020-00501-9.

Zhou, Guanwu, Yulong Zhao, Fangfang Guo, and Wenju Xu (2014). "A Smart High Accuracy Silicon Piezoresistive Pressure Sensor Temperature Compensation System". In: *Sensors* 14.7, pp. 12174–12190. ISSN: 1424-8220. DOI: 10.3390/s140712174.

Zou, Mingxuan, Ye Xu, Jianxiang Jin, Min Chu, and Wenjun Huang (2023). "Accurate Nonlinearity and Temperature Compensation Method for Piezoresistive Pressure Sensors Based on Data Generation". In: *Sensors* 23.13. ISSN: 1424-8220. DOI: 10.3390/s23136167.

Zou, Z., Y. Jin, P. Nevalainen, Y. Huan, J. Heikkonen, and T. Westerlund (Mar. 2019). "Edge and Fog Computing Enabled AI for IoT—An Overview". In: *Proc. IEEE Int. Conf. on Artificial Intelligence Circuits and Systems (AICAS)*. Hsinchu, Taiwan, pp. 51–56.

Appendix A
AI-Assisted Hardware Design with Large Language Models: An Exploratory Study

I.1 Publications Associated with This Chapter

The work presented in this chapter originates from the research activity carried out during my six-month visiting period at Johns Hopkins University, where I investigated the use of generative Artificial Intelligence for hardware design automation. This work directly resulted in the peer–reviewed publication:

- Paola Vitolo et al. (2024d). "Natural Language to Verilog: Design of a Recurrent Spiking Neural Network using Large Language Models and ChatGPT". in: *2024 International Conference on Neuromorphic Systems (ICONS)*, pp. 110–116. DOI: 10.1109/ICONS62911.2024.00024

Readers are referred to this publication for additional technical details and extended experimental results.

I.2 Introduction

The demand for ASICs has been steadily rising, with market forecasts projecting values of up to USD 33.3 billion by 2033 and an annual growth rate of 6.4% ("Application Specific Integrated Circuit Market Full Report" 2024). This trend reflects the growing need for domain-specific hardware in sectors such as IoT, automotive, consumer electronics, and healthcare, often driven by the integration of artificial intelligence and machine learning functionalities ("Application Specific Integrated Circuit Market Full Report" 2024; Sapsanis et al., 2018; Murray et al., 2018; Rogdakis et al., 2024). Yet, the process of designing and verifying efficient custom integrated circuits remains time- and resource-intensive, with ever tighter time-to-market constraints (Y. Li et al., 2018). These challenges are particularly crucial in the context of neuromorphic computing and in-sensor processing, where energy efficiency, latency, and silicon area must be carefully balanced.

This appendix reports an *exploratory* investigation that complements, but does not replace, the constraints-first top-down hardware–software co-design methodology

presented in Chapter II. While Chapters III–V focus on application-driven ISC accelerators and quantitative power/area/latency results, the goal here is to evaluate whether LLMs can act as productivity aids for generating and refining HDL and verification artifacts under designer supervision. For this reason, the results in this appendix should be interpreted primarily in terms of *development workflow* (code generation, modular verification, and iteration), rather than as a new ISC case study.

Generative Artificial Intelligence (GenAI), and in particular Large Language Models (LLMs), has recently emerged as a promising complement to conventional Electronic Design Automation (EDA) flows (Huang et al., 2021). Several studies have shown how LLMs can support hardware design by generating synthesizable hardware description code, assisting in debugging, or even co-developing verification infrastructures (M. Liu et al., 2023; Thakur et al., 2024; Fu et al., 2024; Thakur et al., 2023). Early demonstrations include the design of simple modules such as random number generators with bus interfaces (Meech, 2023) and the use of conversational models to detect and fix design bugs (Fu et al., 2023). GPT models, in particular, have been widely adopted for these tasks (Blocklove et al., 2023; L. J. Wan et al., 2024), enabling the automatic production of Verilog and VHDL with reduced syntax errors. Our previous work also validated the use of GPT-4 to implement a programmable digital spiking neuron array fabricated in SkyWater 130 nm technology (Tomlinson et al., 2024).

Despite these promising results, significant gaps remain in scaling such approaches to more complex neuromorphic architectures. Building on prior work (Tomlinson et al., 2024), this chapter investigates the use of LLMs to design a recurrent spiking neural network (RSNN) in synthesizable Verilog, exploring not only architectural generation but also testbench development and verification. The chosen RSNN extends beyond feedforward or single-layer SNNs by introducing recurrence, thereby enabling the capture of temporal dynamics—an essential feature for processing sequential data in edge and in-sensor scenarios.

To validate the approach, we consider three representative machine learning tasks of increasing complexity: the XOR problem, the IRIS flower classification, and the sequential MNIST benchmark. The LLM-generated Verilog is evaluated both in FPGA prototyping and ASIC synthesis using a fully open-source flow targeting SkyWater 130 nm technology, with a final submission to the Efabless Tiny Tapeout program.

In the broader context of this thesis, this chapter demonstrates how generative AI can accelerate algorithm–hardware co-design, reducing iteration time while supporting the implementation of ultra-low-power neural accelerators for in-sensor computing. The subsequent sections describe the methodology and prompting strategies, the RSNN architecture, validation tasks, implementation results, and a critical discussion of opportunities and challenges in adopting LLM-based design automation within neuromorphic hardware design.

I.3 Methodology and Design Flow

SNNs, regarded as the third generation of artificial neural networks, emulate the spiking behavior of biological neurons. Their event-driven computation leads to more compact and efficient implementations compared to conventional neural networks (Akusok et al., 2019; Cassidy et al., 2013). When extended with recurrent connections

(RSNNs), they are inherently capable of processing temporal sequences and classifying dynamic input streams, making them highly suitable for real-time AI/ML tasks where latency and power are critical constraints.

From an ISC perspective, event-driven spiking computation is also conceptually aligned with energy-proportional operation: when input activity is sparse, computation and memory accesses can be naturally reduced, enabling aggressive duty cycling and fine-grained gating. Therefore, RSNNs provide a useful representative target to explore AI-assisted RTL generation for ultra-low-power neural processing under tight resource constraints.

To realize the proposed RSNN, we adopted a modular, bottom–up design methodology supported by OpenAI's ChatGPT-4. The complex system was decomposed into a hierarchy of smaller, reusable submodules. This hierarchical approach improved manageability, scalability, and reusability, while facilitating early detection of design errors. For each submodule, ChatGPT was prompted to generate synthesizable Verilog code accompanied by inline documentation, which is essential for long-term maintainability and potential extensions. After functional description, ChatGPT was further instructed to produce dedicated testbenches, enabling independent verification of each module.

Once the submodules passed functional verification, they were instantiated together within a top-level module to form the complete RSNN architecture. This systematic integration strategy ensured clear separation of concerns, reduced complexity at each stage, and accelerated development.

All source code and conversation transcripts documenting the design process are made publicly available through the GitHub repository at Andreou-JHULabOrg/ tinytapeout_06_chatgpt_rsnn (Vitolo et al., 2024e).

I.3.1 System Architecture

The synthesized RSNN is depicted in Fig. I.1. The network is organized as a fully connected structure with three spiking inputs and three spiking outputs. Internally, it is arranged in three layers, each comprising three recurrent spiking neurons. Every neuron in a given layer receives input from all neurons in the preceding layer—or directly from the external inputs in the case of the first layer—resulting in three synaptic weights per neuron. Consequently, each layer contains 3×3 connections, leading to a total of $3 \times 3 \times 3 = 27$ weights across the entire network.

The neuron dynamics are described by the leaky integrate-and-fire (LIF) model with recurrence, formulated in Eq. I.1. Here, I_{in} denotes the synaptic input current, U the membrane potential, U_{thr} the firing threshold, S_{out} the output spike, R the reset term, β the decay constant, and V the scaling factor for the recurrent feedback.

$$\begin{aligned} &U(t) > U_{thr} \;\Rightarrow\; S(t+1) = 1, \\ &U(t+1) = \beta U(t) + I_{in}(t+1) + V S_{out}(t) - R U_{thr}. \end{aligned} \tag{I.1}$$

When the membrane potential exceeds the threshold U_{thr}, the neuron generates an output spike S_{out} and triggers the reset signal $R = 1$, which reduces U by subtracting the threshold value. Recurrence is modeled by self-connections in which each neuron reinjects its own spike activity scaled by V. The model also incorporates a

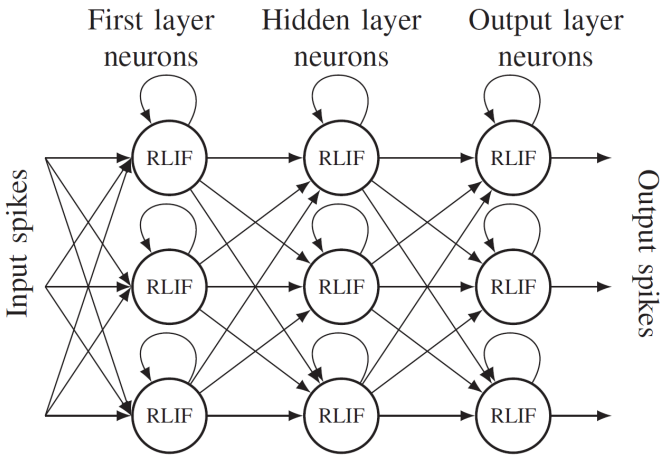

Figure I.1: *Diagram of the desired Recurrent Spiking Neural Network, consisting of 3 fully connected layers, each layer having 3 recurrent spiking neurons. Extracted from (Vitolo et al., 2024d).*

refractory phase, preventing immediate reactivation after a spike and thus improving the biological plausibility of the dynamics.

Following a bottom–up design strategy, the LIF neuron was implemented first. The initial Verilog code produced by ChatGPT underwent iterative refinements: specifications were progressively clarified, and issues were systematically corrected. The final neuron module generated by ChatGPT is reported in Table I.2, alongside its initial draft. Documentation for this block was also automatically produced.

The same iterative process was applied to extend the neuron design with an additional input for the feedback scaling factor. Subsequently, a layer of three neurons was realized, where input currents were generated by multiplying incoming spikes with their corresponding weights. This was followed by the construction of a three-layer network, complemented with a First-In-Parallel-Out (FIPO) memory for parameter storage and a finite state machine (FSM) for memory management. The final top-level system integrates the RSNN layers, memory structures, the FSM-based controller, synchronizers for asynchronous inputs, and registers for spike buffering.

Design constraints required support for both positive and negative weights encoded in two's complement 8-bit fixed-point format, with proper handling of signed overflow and sufficient bit-width extension to ensure reliable accumulation of weighted inputs.

After completing each hardware block, a separate ChatGPT session was initiated with the explicit goal of creating a Verilog testbench. The automatically produced code was iteratively improved: simulation traces were analyzed, errors or omissions were identified, and progressively more precise details were added to the prompts to refine the generated stimuli and checking conditions. In this way, every submodule was validated in isolation before moving on to system-level verification of the integrated top module. Table I.3 provides an example of this process, showing the initial draft and the final version of the LIF neuron testbench produced through iterative refinement.

Listing 1
INITIAL MODULE

```
module LeakyIntegrateFireNeuron(
    input clk,
    input reset_n,
    input [7:0] input_current,
    input [7:0] threshold,
    input [7:0] leak,
    input [3:0] refractory_period,
    output reg fired
);
    // Membrane potential of the neuron
    reg [7:0] membrane_potential;
    // Membrane potential of the neuron
    reg [3:0] refractory_counter;

    always @(posedge clk or negedge reset_n)
    begin
        if (!reset_n) begin
            membrane_potential <= 8'b0;
            refractory_counter <= 4'b0;
            fired <= 1'b0;
        end else begin
            // If in refractory period,
            // decrement the refractory counter
            if (refractory_counter > 0) begin
                refractory_counter <=
    refractory_counter - 1;
            end else begin
                // Update membrane potential
                // with leaky integration
                membrane_potential <=
    membrane_potential + input_current - (
    membrane_potential * leak >> 8);

                // Check if the membrane
                // potential has reached
                // or exceeded the threshold
                if (membrane_potential >=
    threshold) begin
                    // If threshold is
                    // reached, neuron fires
                    fired <= 1'b1;
                    // Reduce membrane
                    // potential by the
                    // threshold value
                    membrane_potential <=
    membrane_potential - threshold;
                    // Set refractory counter
                    // to the specified
                    //refractory period
                    refractory_counter <=
                        refractory_period;
                end else begin
                    // If threshold is not
                    // reached, neuron does
                    // not fire
                    fired <= 1'b0;
                end
            end
        end
    end
endmodule
```

Listing 2
FINAL MODULE

```
module LeakyIntegrateFireNeuron(
    input clk,
    input reset,
    input enable,
    input [7:0] input_current,
    input [7:0] threshold,
    input [7:0] decay,
    input [7:0] refractory_period,
    output reg spike_out
);

    reg [7:0] membrane_potential = 8'b0;
    reg [7:0] refractory_counter = 8'b0;
    wire signed [9:0] potential_update;

    // Calculate the potential update in signed
    arithmetic based on input current and decay
    assign potential_update = $signed({
    membrane_potential[7], membrane_potential
    [7], membrane_potential}) +$signed({
    input_current[7], input_current[7],
    input_current}) +(membrane_potential[7] ?
    $signed({decay[7], decay[7], decay}) : -
    $signed({decay[7], decay[7], decay}));

    always @(posedge clk or posedge reset) begin
        spike_out <= 1'b0;
        if (reset) begin
        membrane_potential <= 8'b0;
        refractory_counter <= 8'b0;
        end else if (enable) begin
        if (refractory_counter > 0) begin
        refractory_counter <= refractory_counter
    - 1;
        end else begin
        // Check for potential overflow
        if (potential_update[9] &&
    potential_update < -128)
        // If overflow in the negative direction,
        // saturate to minimum value
        membrane_potential <= 8'b1000_0000;
        else if (potential_update > 127)
        // If overflow in the positive direction,
        // saturate to maximum value
        membrane_potential <= 8'b0111_1111;
        else
        // If no overflow, assign the
        // 8 LSBs of updated potential
        membrane_potential <= potential_update
    [7:0];
        // Check if membrane potential exceeds
        // the threshold
        if ($signed(membrane_potential) >=
    $signed(threshold)) begin
        // Neuron fires
        spike_out <= 1'b1;
        // Reduce membrane potential
        // by the threshold value
        membrane_potential <= $signed(
    membrane_potential) - $signed(threshold);
        // Set refractory counter to the
        // specified refractory period
        refractory_counter <= refractory_period;
        end
        end
        end
    end
endmodule
```

Figure I.2: *Leaky Integrate-and-Fire Spiking Neuron Verilog Module generated by ChatGPT. Extracted from (Vitolo et al., 2024d).*

Listing 3
INITIAL MODULE

```
`timescale 1ns / 1ps
module tb_LeakyIntegrateFireNeuron;
    reg clk;
    reg reset;
    reg [7:0] input_current;
    reg [7:0] threshold;
    reg [7:0] decay;
    reg [7:0] refractory_period;
    wire spike_out;
    LeakyIntegrateFireNeuron uut (
        .clk(clk),
        .reset(reset),
        .input_current(input_current),
        .threshold(threshold),
        .decay(decay),
        .refractory_period(refractory_period),
        .spike_out(spike_out)
    );
    initial begin
        clk = 0;
        forever #10 clk = !clk;
    end
    initial begin
        reset = 1;
        input_current = 0;
        threshold = 50;
        decay = 5;
        refractory_period = 5;
        #25 reset = 0;
        // Stimulus 1:
        // Small input current, no spike
        input_current = 10;
        #40;
        // Stimulus 2: Larger current
        // to trigger spike
        input_current = 60;
        #20;
        // Stimulus 3: Check refractory
        // behavior, should not spike
        // despite high current
        input_current = 60;
        #100;
        // Stimulus 4: After refractory
        // period, should be able
        // to spike again
        input_current = 60;
        #100;
        // Complete the test
        #50 $finish;
    end
    initial begin
        $monitor("Time = %t, Input Current = %d,
    Membrane Potential = %d, Spike = %b", $time,
     input_current, uut.membrane_potential,
    spike_out);
    end
endmodule
```

Listing 4
FINAL MODULE

```
`timescale 1ns / 1ps
module tb_LeakyIntegrateFireNeuron;
    reg clk;
    reg reset;
    reg enable;
    reg [7:0] input_current;
    reg [7:0] threshold;
    reg [7:0] decay;
    reg [7:0] refractory_period;
    wire spike_out;
    LeakyIntegrateFireNeuron uut (.clk(clk),.
    reset(reset), .enable(enable),.input_current
     (input_current),.threshold(threshold), .
    decay(decay), .refractory_period(
    refractory_period),.spike_out(spike_out));
    initial begin
        clk = 0;
        forever #10 clk = !clk;
    end
    initial begin
        reset = 1; enable = 0;
        input_current = 0; threshold = 127;
        decay = 1; refractory_period = 10;
        #25 reset = 0; enable = 1;
        input_current = 10;  #40;
        input_current = 60;  #20;
        input_current = 60;  #100;
        enable = 0;
        input_current = 60;  #40;
        enable = 1;          #20;
        input_current = 30;  #100;
        enable = 0;          #40;
        enable = 1;
        input_current = 75;  #60;
        reset = 1;           #30;
        reset = 0;           #40;
        input_current = 15;  #100;
        enable = 0;          #40;
        enable = 1;          #60;
        input_current = 5;
        $display("Starting test with low input
    and high threshold at time %t", $time);
        fork
            begin
                wait (spike_out == 1);
                $display("Spike occurred at time
    %t with low input current", $time);
                $finish;
            end
            begin
                #10000;
                $display("No spike after 10000ns
    with low input current");
                $finish;
            end
        join
    end
    initial begin
        $monitor("Time = %t, Input Current = %d,
    Membrane Potential = %d, Spike = %b", $time,
     input_current, uut.membrane_potential,
    spike_out);
    end
endmodule
```

Figure I.3: *Test Bench for the LIF Spiking Neuron Verilog Module generated by ChatGPT. Extracted from (Vitolo et al., 2024d).*

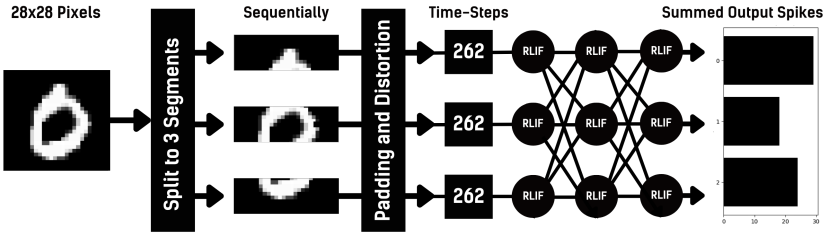

Figure I.4: *Block Diagram describing the pipeline used for our proposed Sequential MNIST model. Extracted from (Vitolo et al., 2024d).*

I.4 Validation on Benchmark Tasks

To evaluate the functional correctness of the Verilog modules generated with ChatGPT, the architecture was tested on three representative case studies: the exclusive OR function, the IRIS flower dataset, and the Sequential MNIST digit recognition task. These benchmarks were selected to progressively increase the level of complexity, from simple nonlinear decision boundaries to multi-class classification and finally to sequential, temporally extended inputs.

The behavioral modeling of the recurrent leaky integrate-and-fire (RLIF) neurons with one-to-one recurrent connections was performed using SNNTorch (Eshraghian et al., 2023), complemented by the available documentation for this neuron type. Network weights and parameters were quantized with the support of SNNTorch (Eshraghian et al., 2023) and Brevitas (Pappalardo et al., 2024), ensuring consistency between the software model and the hardware implementation. The tested network consisted of three layers, each realized as a quantized linear block of three RLIF neurons with recurrent feedback.

I.4.1 Case Study 1: Exclusive OR

The exclusive OR (XOR) problem represents a minimal nonlinear classification task and was used as a first validation of the synthesized RSNN. In this setup, only two out of the three input neurons and two out of the three output neurons were utilized. Despite its simplicity, the task confirmed the non-linear separability of the network, with classification accuracy reaching up to 95%.

I.4.2 Case Study 2: IRIS Flower Classification

The IRIS dataset provides a classical benchmark for evaluating multi-class classification. It involves distinguishing among three flower species: *setosa*, *versicolor*, and *virginica*. To adapt the dataset to the three-input architecture, the last feature was excluded, resulting in a three-dimensional input vector. This reduction allowed direct mapping to the network inputs while preserving the class structure. Using this configuration, the RSNN achieved accuracies of up to 96.6%, demonstrating the suitability

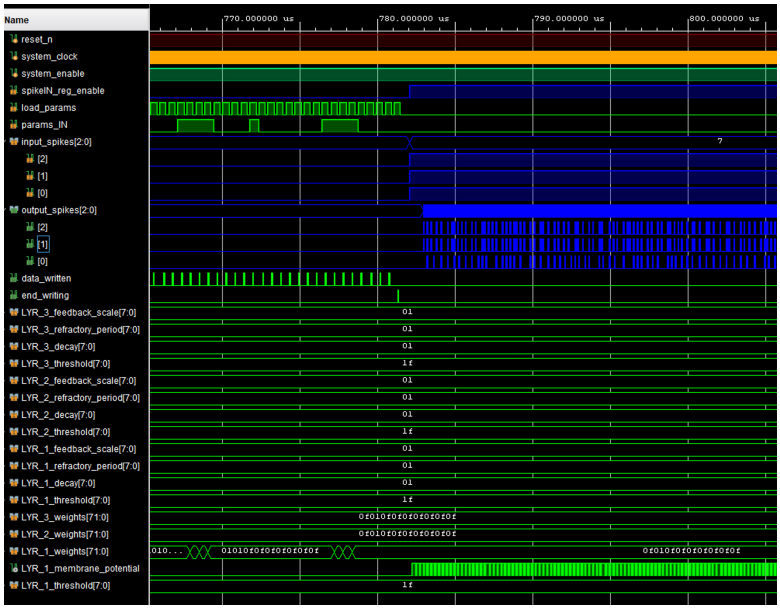

Figure I.5: *Post-implementation timing simulation of the RSNN hardware design generated with ChatGPT on Spartan-7 FPGA. Two operating phases can be observed: startup (parameter loading) and running (spike processing). Extracted from (Vitolo et al., 2024d).*

of the design for structured multi-class classification problems.

I.4.3 Case Study 3: Sequential MNIST Digit Recognition

While the XOR and IRIS experiments confirmed classification capability, they did not fully exploit the temporal processing strength of recurrent spiking architectures. To this end, the Sequential MNIST benchmark was adopted, with and without enabling recurrence. For this study, only the first three digit classes (0, 1, and 2) were considered, as illustrated in Fig. I.4. Each MNIST image was divided into three equal vertical segments (top, middle, bottom) and then mapped to the three input neurons of the RSNN. To achieve equal segmentation, the images were distorted with padding, reducing the number of timesteps from 784 to 262.

The results confirmed that the recurrent dynamics improved performance in sequential classification tasks. For the three-class classification, the RSNN reached an accuracy of 89%, while binary classification (two classes at a time) achieved up to 94.7%. This case study highlights the advantages of incorporating recurrence in spiking neural networks for processing temporal or segmented inputs.

I.5 Results and Discussion

The iterative design sessions with ChatGPT successfully produced synthesizable Verilog code for all the RSNN modules after a total of 117 iterations. Table I.1 summarizes the conversational process, highlighting the number of iterations, code size,

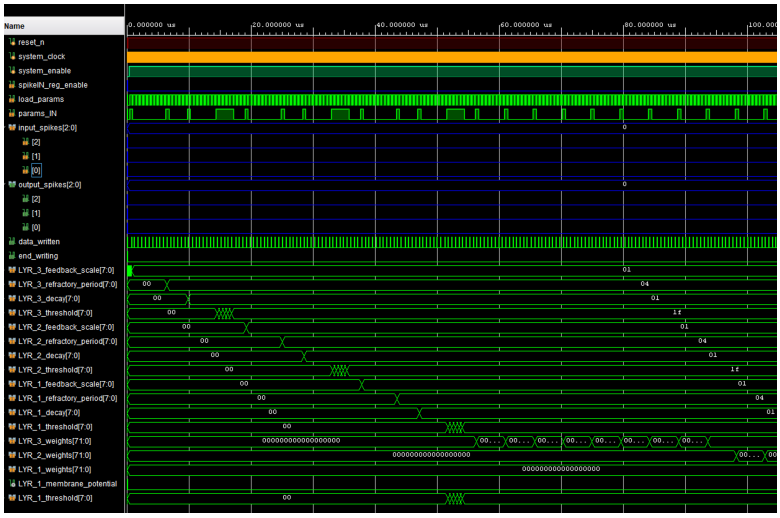

Figure I.6: *Zoomed-in timing simulation of the startup phase. The load_params signal is asserted, enabling the loading of network parameters into memory before the system transitions to the running mode. Extracted from (Vitolo et al., 2024d).*

and the main refinements required. Among the modules, the LIF neuron and the RLIF layer proved to be the most demanding, requiring 38 and 17 iterations respectively across multiple conversations.

Two aspects mainly contributed to the higher number of iterations: (i) the complexity of handling overflow and underflow conditions with proper bitwidth extension for current accumulation, and (ii) the size of the codebase. Smaller modules with fewer lines of code were generally easier to debug and converge more quickly, even when overflow management was required. These findings emphasize the value of a modular design flow supported by clear, well-defined requirements and illustrative examples.

I.5.1 FPGA Prototyping

The generated design was prototyped on a Digilent CMOD S7 development board equipped with a Xilinx Spartan-7 FPGA, using the Xilinx Vivado Design Suite. Post-implementation analysis reported a utilization of 1011 LUTs and 507 FFs, with a maximum operating frequency of 83 MHz. Power analysis was performed through Switching Activity Interchange Format (SAIF)-based simulations, yielding a total estimated consumption of 65 mW, of which 4 mW was dynamic and 61 mW static. Timing simulations (Figs. I.5 and I.6) revealed two functional phases: a *startup phase*, where network parameters are loaded into memory, and a *running phase*, during which input spikes are processed. This confirms the correctness of memory initialization and real-time spike processing.

Table I.1: *Summary of Verilog Code Generation with ChatGPT. Extracted from (Vitolo et al., 2024d).*

Module Name	**Chat #**	**Iterations #**	**Lines of Code**	**Main Refinements**
LIF Neuron	2	38 (24+14)	33	Overflow/underflow handling Bitwidth adjustment Sign extension Syntax best practices Additional input signals
RLIF Neuron	1	6	19	Overflow/underflow handling Bitwidth adjustments Sign extension
RLIF Layer	2	17 (10+7)	63	Clarification of requirements Input data formatting Parameter sharing Syntax refinements Overflow management
RSNN	1	8	37	Clarification of behaviour Verilog best practices
FIPO Memory	1	12	21	Clarification of behaviour Control signal insertion
RegN	1	5	6	Parameterization Control signals Reset behaviour
Control Memory	1	9	55	Clarification of behaviour Syntax corrections
Top Module	1	22	103	Clarification of behaviour Connection refinement

I.5.2 ASIC Implementation and Tiny Tapeout

For ASIC validation, the design was synthesized and implemented using the SkyWater 130 nm open PDK through the OpenLane flow. OpenLane automates RTL-to-GDSII design, including logic synthesis, placement, routing, and signoff checks. The final layout is shown in Fig. I.7. The implementation achieved an area occupation of 0.11 mm^2 with 5187 cells, distributed as follows: 1635 combinational logic cells, 1577 taps, 647 NORs, 580 ORs, 512 flip-flops, 495 buffers, 345 NANDs, 337 ANDs, 242 miscellaneous cells, 220 multiplexers, 124 inverters, and 49 diodes.

All signoff checks were passed successfully, confirming the robustness of the design flow and the viability of ChatGPT-assisted hardware generation for ASIC targets. These results demonstrate the potential of LLM-based design assistance as a valuable complement to traditional hardware development processes.

I.5.3 Limitations, auditability, and explainability

This exploratory study has three main limitations.

- **Limited quantitative assessment of productivity impact.** While Table I.1 reports the number of iterations and refinements required to converge to synthesizable modules, we do not claim a generalizable speedup factor with respect to a conventional manual flow. A rigorous assessment would require controlled comparisons (same specification, multiple designers, time-to-convergence and bug-rate metrics).
- **Bottom–up demonstration versus full constraints-first co-design.** The goal here is to evaluate LLM assistance for RTL and verification artifact generation. Integrating this capability into the constraints-first co-design methodology of Chapter II would require starting from an application-driven specification (e.g., one of the ISC case studies) and quantifying the end-to-end impact on design iterations, verification effort, and implementation closure.
- **Explainability and rationale of LLM-generated design choices.** The present study relies on transparent prompting and on conventional verification/EDA feedback as the main correctness filters. Dedicated LLM explainability tools (e.g., systematic rationale extraction, prompt ablations, or structured auditing of decision paths) were not applied, and represent an important direction for future work. To support reproducibility and post-hoc inspection, all prompts and conversation transcripts are made available in the associated public repository, enabling independent auditing of the design process.

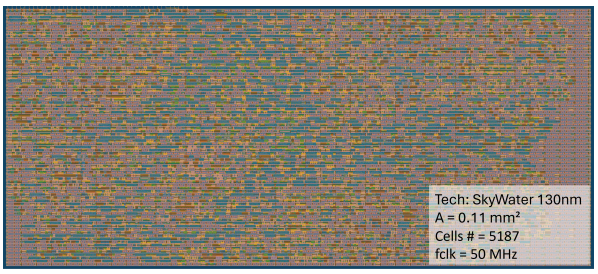

Figure I.7: *Final GDSII layout of the RSNN hardware design generated by ChatGPT using the SkyWater 130 nm PDK. Extracted from (Vitolo et al., 2024d).*

Appendix B
Extended Acknowledgments

I feel incredibly fortunate for all the people I met during these Ph.D. years, who enriched me both professionally and personally. It is often said that the most important choice is not the Ph.D. topic itself, but the laboratory and supervisors you work with. I could not agree more, and I was truly lucky on every side.

First and foremost, I wish to thank my supervisor Prof. Gian Domenico Licciardo for his constant guidance, for treating me as a colleague with whom to reason and discuss ideas, and for introducing me to both academic and industrial research with integrity and enthusiasm, always finding time for me even when there was no time. I would not even have started this journey without him. Thank you for continually encouraging and motivating me to do my best. I am also grateful to Prof. Alfredo Rubino, who was always supportive and made this doctoral path possible.

My sincere thanks go to the researchers Rosalba and Luigi, now a professor, for making me feel part of the lab from day one, always ready to help with technical questions as well as to listen when personal challenges arose.

To my fellow Ph.D. colleagues Fiorita, Andrea ”Grande”, Nicola, and Giuseppe, thank you for the shared anxieties, late-night laughs, chocolate breaks, coffees, and pizza nights that made long days joyful.

I am grateful to my thesis students—now dear friends—such as Mariarosaria, Sara, Antonio, Ilenia, Pio and all the others, for the trust they placed in me. It has been a privilege to serve as your co-supervisor, and our collaboration has truly been a two-way exchange: discussing ideas and advancing each project together has broadened my perspective as much as, I hope, it has yours.

I also warmly thank Federica, with whom—together with Mariarosaria and Luiss—I shared many lovely moments, from pizzas out to long chats and coffee breaks.

To Anna Chiara, with whom I shared the moments before and at the very beginning of the Ph.D. journey. She witnessed my early decisions and supported me during my later time in Milan, through both good and challenging days.

Thanks to the guys of the IEEE Student Branch of Salerno-Nic, Rosario, Vincenzo, Valter. Being part of the branch was a wonderful experience of teamwork and enthusiasm. It was great fun to join, explore that world, and organize events together, always finding time even when our schedules seemed impossibly full.

I would like to thank all the professionals I was fortunate to meet, each of whom gave me far more than was ever required.

First and foremost, my supervisor at STMicroelectronics, Danilo Pau, whose experience and enthusiasm for his work were truly inspiring. He offered countless suggestions on how to become a better researcher, including the invaluable habit of keeping a personal library of scientific papers and reading at least one each day, on any topic, to add to that collection.

I feel especially lucky to have met such wonderful colleagues in the Milan hardware team at STMicroelectronics—among them Carmine, Michele, and my former master's thesis co-advisor Antonio—who made me feel at home even "up North" and were friends as well as colleagues, always ready to help with both personal and technical matters. I am grateful to Thomas, Giuseppe, and Tommaso at STMicroelectronics, whom I deeply admire for their exceptional technical skills—spanning hardware, machine learning, and much more—as well as for their remarkable personal qualities.

A heartfelt thanks to Simone Zezza, who guided me through the entire back-end of my chip design in Milan, patiently teaching me so much and sharing his passion for his work, always finding time for me despite an extremely busy schedule.

I am grateful to the RFI team, especially Giuseppe, who always found time to explain everything to me, first providing an overall view and then guiding me through every single module. He offered advice not only on technical matters but also on professional and personal aspects of life, and I truly appreciated his constant availability and kindness. My thanks also go to Giovanni, with whom I greatly enjoyed working and collaborating.

I warmly thank the friends I met overseas who made me feel welcome and included. First of all, Heather, with whom I traveled across half of the United States—one of the most memorable journeys I have ever taken. Thank you for being not only a mentor to me but also a friend with whom I could share a passion for running and for sipping tea while watching the sunset.

I am grateful to the wonderful guys met at Johns Hopkins Lab—George, Hee Young, Akwasi, and Daniel—who made me feel part of the group and created many unforgettable moments.

A special thanks goes to Prof. Andreas for believing in me and giving me the opportunity to have one of the most rewarding experiences of my Ph.D. in his laboratory at Johns Hopkins, as well as for the encouraging messages he sent during my family's difficult times.

In this context, I also wish to express my deepest gratitude to all the doctors and nurses who, in these past months, have cared for my father and have supported and guided to me as well. I would not be writing this thesis without you.

Special thanks to my Ph.D. coordinator, Prof. Massimo De Santo, who gave me the final encouragement to complete this journey and was always available whenever I encountered any difficulties during the doctoral program.

I am grateful to all the other professors and colleagues I have been fortunate to meet, each of whom has given me so much. From the professors at the University of Salerno, who have watched me grow since my very first year as an undergraduate, to all the professors and colleagues I encountered during doctoral courses, conferences, and workshops, every interaction has enriched my academic journey and broadened my perspective.

Special thanks go to Akwasi, who worked with me on the staff of the Telluride workshop and with whom I shared the trips to Montrose to prepare for the event. I

am also especially grateful to Giorgia, Luna, Rej, Giulia, and Professors Shihab, Rajkumar, and Ernst, whom I was fortunate to meet there, for all the wonderful and challenging moments we shared—from the magical nights watching the Milky Way above Telluride and our many gondola rides, to late-night project sessions, Sunday hikes with overnight camping, laughter, and valuable professional and personal advice. Those were intense days, and you truly became my family there.

Finally, I want to thank my family and friends for always being there for me, even when I was too busy, too exhausted, too stressed, or simply "too much".

To my sister Antonella and my brother-in-law Alfonso, thank you for being present even for a quick five-minute visit or a short outing. I am grateful for the way you watched over me, kept me going with home-cooked meals and treats, and offered the little distractions that helped me through the toughest moments.

I also thank my sister Luisa, my brother-in-law Gerardo, and my beloved nieces and nephews Leonardo, Lorenzo, and Lucia, who—even from a distance—have always been there for me with advice or simply for a friendly chat.

I am also deeply grateful to my other family—Sara, Gianpiero, Luciana, and Alfonso. With each of you there have been times when we were close and times when we were apart, whether because of busy schedules or physical distance, yet whenever we meet it feels as if no time has passed. Thank you, Luciana, for our long conversations, sharing both difficult moments and happy ones, and to you and Alfonso for caring about me as Antonella and Alfonso do, making sure I was eating and helping me take a break when I needed it. I would also like to warmly thank Mrs. Lina, whose kindness, thoughtful messages, and patient listening have always made me feel welcome and supported.

I thank my dear friend Dina, whom I have known since the first year of high school. You know everything about me—my anxieties, fears, joys, and insecurities—whether I speak them aloud or not, and even when we were physically apart it never felt like we were distant. You are the person who immediately understands why I am silent or why I need to talk, and who instinctively senses how I am feeling.

I also thank Simone, with whom I shared my entire undergraduate and master's journey and with whom I continue to exchange thoughts on every professional and personal aspect. You have always given me space to vent, a smile when I could only cry, and unwavering support when I felt I could not go on.

Finally, but no less important, I want to thank my life partner, Aniello. We have been together for almost thirteen years, growing side by side, sometimes drifting apart and finding our way back, sharing both highs and lows. Even in our lowest moments, when distance or disagreements arose, you were always truly there for me—supporting every choice I made and believing in me even when I doubted myself. Despite your demanding work schedule, you always found a way to reach out, even just keeping the phone line open while you worked so that I would not feel alone. You are the person I instinctively call first—whether in moments of need or to share exciting news and successes. Your quiet, steadfast presence has been an anchor throughout this journey, and I am endlessly grateful for your love and unwavering support.

All of you have been by my side through every difficult moment and every important decision, sharing not only my worries but also my joys, successes, and setbacks.

Chapter B

This milestone is not mine alone. It belongs to all of you who have shared this path with me, in big ways and small, and whose kindness, wisdom, and friendship have shaped both my research and my life.