DOTTORATO DI RICERCA IN

DATA SCIENCE AND COMPUTATION

Ciclo 35

Settore Concorsuale: 09/E3 - ELETTRONICA

Settore Scientifico Disciplinare: ING-INF/01 - ELETTRONICA

ENERGY-EFFICIENT TIME SERIES ANALYSIS WITH MACHINE LEARNING AND DEEP LEARNING ON EMBEDDED COMPUTING PLATFORMS

Presentata da: Marcello Zanghieri

Coordinatore Dottorato

Daniele Bonacorsi

Supervisore

Luca Benini

Co-supervisori

Simone Benatti

Francesco Conti

Esame finale anno 2024

ALMA MATER STUDIORUM - UNIVERSITY OF BOLOGNA

Energy-Efficient Time Series Analysis with Machine Learning and Deep Learning on Embedded Computing Platforms

by

Marcello Zanghieri

A thesis submitted for the degree of
Doctor of Philosophy

in the
School of Engineering and Architecture
Department of Electrical, Electronic, and Information Engineering (DEI)

May 2024

I dedicate this thesis to my family.

Acknowledgments

I wish to thank my supervisor, Prof. Luca Benini, for the opportunity to pursue my PhD in his research group and for his constant guidance. I also thank my co-supervisors, Prof. Simone Benatti and Prof. Francesco Conti, for their persevering supervision, their teachings, and the motivation they gave me. I thank all three of them for the support and independence they gave me. I also want to thank Dr. Francesco Beneventi for the technical help at the beginning of my PhD and for his vast patience.

I thank Prof. Giacomo Indiveri for welcoming me to his NCS group at INI for a visiting research period. I thank him and Dr. Elisa Donati for following my work.

I thank Dr. Elisabetta Farella and Prof. Melika Payvand for their willingness to review this thesis.

I address special thanks to Panagiota (Iota) Dimopoulou for guiding us DSC PhD students through the bureaucratic and formal stages of the programme in the first years. I also thank our PhD coordinator, Prof. Daniele Bonacorsi, for the accurate and timely directions at the end of the PhD.

Finally, I want to thank all my colleagues at the EEES Lab for creating this enjoyable and inspiring environment.

Abstract

The present Ph.D. thesis presents techniques and solutions for energy-efficient time-series analysis based on automated learning executed on resource-constrained, low-power computing platforms, with an interest in both Deep Learning and traditional, non-deep Machine Learning. This dissertation spans diverse domains, from algorithmic research on the accuracy-efficiency tradeoff in processing different biosignals to applied research inspired by industrial scenarios.

The unifying methodology that brings all the research questions addressed in this thesis under the same perspective is the interest in time-series analysis as a task to be performed in the presence of the resource constraints characteristic of low-power edge computing devices. In particular, a special focus is devoted to single- and multi-core embedded microcontrollers (MCUs).

This dissertation covers the three major types of automated learning tasks: binary classification, multi-class (single-label) classification, and regression. Starting from binary classification, this work presents a proximity sensor for active safety in industrial machinery, accurate and robust against acoustic noise, and a setup for epilepsy detection from intracranial electroencephalography. Both solutions are based on a Temporal Convolutional Network (TCN) executed on an embedded MCU, showcasing the power and versatility of the approach. Moving to multi-class (single-label) classification and regression, the research effort was entirely devoted to the topic of hand modeling from the surface electromyographic (sEMG) signal. Starting with off-device TCNs for the recognition of discrete hand gestures, the classification setup is advanced by carrying out the deployment on a multi-core MCU and by studying heuristics for unsupervised adaptation to compensate for changes in arm posture. Then, regression was addressed for a more fluid and versatile control of Human-Machine Interfaces (HMIs). After developing an embedded TCN accurate in hand kinematics estimation, I addressed the modeling of both hand kinematics and force with event-based features, which are computationally cheaper and promising for future porting onto event-driven devices with reduced latency and energy consumption. The contributions in sEMG-based hand modeling advance the field of non-invasive intuitive wearable HMIs.

As a whole, this research work proves the success of the embedded approach to time-series Machine Learning, achieving SoA accuracy and efficiency and thus proving promising for impactful applications in the industrial, clinical, and consumer domains.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, the supervised analysis of time series data by means of Machine Learning (ML) and Deep Learning (DL) [1] involves a wide variety of application scenarios, such as healthcare [2], biosignals processing [3]–[6], human activity recognition [7], [8] acoustic recognition [9], cybersecurity, and structural health monitoring and predictive maintenance in Industry 4.0 [10], [11]. On the one hand, **time series classification** has the purpose of developing automated learning models to infer a categorical class label starting from the input data [1]. On the other hand, **time series regression** [12], [13] aims to estimate a numerical output rather than a categorical one. It is worth remarking that regression is not a synonym of forecasting: time series forecasting is a subset of time series regression, which is the more general task of modeling the relationship between temporal data and one or more external variables, also in the present.

In the years following the recent DL revolution, the Recurrent Neural Networks (RNNs) [14] represented the State-of-the-Art category of models for time series analysis based on automated learning. More recently, Temporal Convolutional Networks (TCNs) [15], [16] and attention-based Transformers [17] have outperformed RNNs. The research presented in this dissertation focuses on TCNs, with the perspective of algorithmic deployment onto resource-constrained embedded devices, especially Microcontroller Units (MCUs). Although Transformers are more recent, it is worth noting that the recent disruptive AI applications based on Transformers are built on models such as the Bidirectional Encoder Representations from Transformers (BERT) [18] and the Generative Pre-trained Transformer (GPT) [19] (especially GPT-3, GPT-3.5, and GPT-4), which are constituted by up to billions of parameters. Thus, the breakthroughs in AI relying on these large Transformers have yet to be ported to the edge or embedded domain, where both Transformers and TCNs are worth the ongoing active research endeavors.

TCNs are a category of Convolutional Neural Networks (CNNs) specialized for time series thanks to 1-dimensional convolutions performed in the time dimension of the input. TCNs combine an accuracy comparable to or better than RNNs with the addition of computational advantages in terms of reduced memory requirements, higher arithmetic intensity, and more data reuse [15], [16]. Hence, TCNs are promising for embedded computing where inference is executed on Internet of Things (IoT) edge devices, in contrast with the centralized cloud computing servers. Porting TCNs to the edge is enabled by the engineering effort toward optimization to make them affordable for the strict memory and energy budgets. This porting typically involves exploring the tradeoff between accuracy and computational efficiency to improve the Pareto frontier and select the desired working point.

This thesis addresses the aforementioned tradeoff by validating, deploying, and profiling both TCNs and methods relying on event-based feature extraction, comparing the performance of the two approaches as to accuracy and execution at the edge. In general, developing efficient embedded ML/DL applications requires **hardware-software co-design**, i.e., a vertical approach that encompasses several layers from the level of processor design to the level of exploration, selection, and optimization of algorithms. The work presented in this thesis focuses on the level of algorithm exploration and profiling for specific applications, also involving the curation of novel task-specific datasets. As to the lower engineering levels of embedded ML/DL, the work exploited existing State-of-the-Art (SoA) frameworks and tools. Combining these two levels has yielded SoA results in accuracy and efficiency in the targeted applications. These contributions are exposed in detail in the next section.

1.1 Contributions & Thesis Structure

The contribution of the research presented in this thesis spans the three main categories of tasks for automated learning: binary classification, multi-class (single-label) classification, and regression. This progression allowed to test the edge-oriented methodology for time-series ML/DL on tasks of increasing pattern complexity. All the research problems addressed have in common the time-series nature of the data, as well as the central role of microcontrollers as target platforms for the developed models.

- **Binary classification**

  The contributions in binary classification involve two domains: the development of ultrasound-based proximity sensor for active safety in industrial machinery and the clinical topic of automatized data-driven epilepsy detection.

– **TCN-based low-latency collision-avoidance safety system for industrial machinery.** Modern manufacturing relies on complex machinery requiring skills, attention, and precise safety certifications. Protecting operators in the machine's surroundings while at the same time reducing the impact on the normal workflow is a major challenge. In particular, safety systems based on proximity sensing of humans or obstacles require that the detection is accurate, low-latency, and robust against variations in environmental conditions. I present (3.1) a functional safety solution for collision avoidance relying on Ultrasounds (US) and a TCN suitable for deployment directly at the edge on a low-power MCU. The setup allowed to acquire a sensor-fusion dataset with 9 US sensors mounted on a real industrial woodworking machine. Applying incremental training, the presented TCN achieved sensitivity 90.5%, specificity 95.2%, and AUROC 0.972 on data affected by the typical acoustic noise of an industrial facility, an accuracy comparable with the SoA. Deployment on an STM32H7 MCU yielded a memory footprint of 560 B ($3\times$ less than SoA), with an extremely low latency of 5.0 ms and an energy consumption of 8.2 mJ per inference (both $> 2.3\times$ less than SoA). The presented solution increases its robustness against acoustic noise by leveraging new data, and it fits the resource budget of real-time operation execution on resource-constrained embedded devices. It is thus promising for generalization to different industrial settings and for scale-up to wider monitored spaces.

– **Low-latency epilepsy detection from iEEG with a TCN.** Epilepsy is a severe neurological disorder that affects about 1% of the world population, and one-third of cases are drug-resistant. Apart from surgery, drug-resistant patients can benefit from closed-loop brain stimulation, eliminating or mitigating the epileptic symptoms. For the closed-loop to be accurate and safe, it is paramount to couple stimulation with a detection system able to recognize seizure onset with high sensitivity and specificity and short latency while meeting the strict computation and energy constraints of always-on real-time monitoring platforms. I present (3.2) a novel setup for iEEG-based epilepsy detection, exploiting a TCN optimized for deployability on low-power edge devices for real-time monitoring. The approach is tested on the Short-Term SWEC-ETHZ iEEG Database, containing a total of 100 epileptic seizures from 16 patients (from 2 to 14 per patient) comparing it with the SoA approach, represented by Hyper-Dimensional Computing (HD). The TCN attains a detection delay which is 10 s better than SoA, without a performance drop in sensitivity and specificity. Contrary to previous literature, this setup enforces a time-consistent setup, where training seizures always precede testing seizures chronologically. When deployed on a commercial low-power parallel MCU, each inference with the model has a latency of only 5.68 ms and an energy cost of only 124.5 µJ if executed on 1 core, and latency 1.46 ms and an energy cost 51.2 µJ if parallelized on 8 cores. This latency and this energy consumption, lower than the current SoA, demonstrate the solution's suitability for real-time, long-term embedded epilepsy monitoring.

- **Classification: sEMG-based hand gesture recognition**

  The contribution in multi-class (single-label) classification belongs entirely to the topic of sEMG-based hand gesture recognition. In particular, I addressed the sEMG inter-posture, inter-session, and inter-day variability by first developing a non-deployed TCN accurate in inter-day scenarios; then, I developed a TCN that is both accurate on inter-day scenarios (on a novel 20-session dataset) and fully deployed onto a parallel ultra-low-power MCU; finally, I developed an unsupervised heuristic for online adaptation to counter changes in arm posture. These contributions successfully target the issue of sEMG inherent variability, which is currently one of the main obstacles to the sEMG-based control of HMIs.

  - **Temporal variability analysis in sEMG hand grasp recognition with a TCN.** I present (4.1) a TCN-based approach that improves by 7.6% the best results in the literature on the NinaPro DB6, a reference dataset for temporal variability analysis of sEMG. Moreover, when targeting the much more challenging inter-session accuracy objective, the method achieves an accuracy drop of just 4.8% between intra- and inter-session validation. This proves the suitability of the setup for a robust, reliable, long-term implementation. Furthermore, the network is distilled using deep network quantization and pruning techniques, demonstrating that the approach can use down to 120× lower memory footprint than the initial network and 4× lower memory footprint than a baseline Support Vector Machine, with an inter-session accuracy degradation of only 2.5%, proving that the solution is suitable for embedded resource-constrained implementations.
  - **Robust real-time embedded sEMG recognition framework with a TCNs.** I present (4.2) a complete wearable-class embedded system for robust sEMG-based gesture recognition based on TCNs. Firstly, a novel TCN topology (TEMPONet) is developed and tested on a benchmark dataset (Ninapro), achieving 49.6% average accuracy, 7.8% better than the current SoA. Moreover, an energy-efficient embedded platform is designed based on GAP8,

  a novel 8-core IoT processor. Using this embedded platform, a second 20-session dataset is collected to validate the system on a setup representative of the final deployment. The recognition achieves 93.7% average accuracy with the TCN, comparable with a SoA SVM approach (91.1%). Finally, the analysis profiled the performance of the network implemented on GAP8 by using an 8-bit quantization strategy to fit the memory constraint of the processor. This deployed application reaches a 4× lower memory footprint (460 kB) with a performance degradation of only 3% accuracy. The execution profiled on the GAP8 platform shows that the quantized network executes a single classification in 12.84 ms with a power envelope of 0.9 mJ, making it suitable for a long-lifetime wearable deployment.

  – **Online unsupervised arm posture adaptation for sEMG-based gesture recognition.** I present (4.3) an unsupervised adaptation technique for sEMG classification and apply it to arm posture variability. The approach relies on aligning the Principal Components (PCs) of new data with the PCs of the training set. No classifier retraining is required, and the PCs are estimated online, consuming one sample at a time without storing any data. The method is validated on the UniBo-INAIL dataset, showing that it recovers 37% to 51% of the inter-posture accuracy drop. The solution is deployed on GAP9, a parallel ultra-low-power microcontroller, obtaining a latency within 3.57 ms and an energy consumption within 0.125 mJ per update step. These values satisfy the constraints for real-time operation on embedded devices. This solution is unsupervised and thus suitable for real-world incremental learning conditions where ground truth is not available.

- **Regression: sEMG-based estimation of hand kinematics and force**

  Also for automated-learning regression tasks, the contribution of this thesis is entirely dedicated to sEMG-based control policies oriented to HMIs. This research effort aims at a more natural, fluid, and intuitive control thanks to the estimation of hand joint angles and simultaneous finger forces. Compared to the work sEMG classification in the previous part, an additional advance resides in exploring event-based features and regression pipelines that are computationally cheaper than DNNs and more amenable to deployment on event-driven computing platforms, which can yield reduced latency and energy consumption in future research work.

  – **sEMG-based regression of hand kinematics with TCNs.** I present (5.1) a regression framework based on TEMPONet (4.2), a SoA TCN for sEMG decoding, which is further optimized for deployment. The approach is tested on the NinaPro DB8 dataset, targeting the estimation of 5 continuous degrees of freedom for 12 subjects (10 able-bodied and 2 trans-radial amputees) performing a set of 9 contralateral movements. The TCN achieves a Mean Absolute Error (MAE) of 6.89°, which is 0.15° better than the SoA. The model reaches this accuracy with a memory footprint of only 70.9 kB, thanks to `int8` quantization. This is remarkable since high-accuracy SoA neural networks for sEMG can reach sizes up to tens of MB if deployment-oriented reductions like quantization or pruning are not applied. The model is deployed on the GAP8 edge microcontroller, obtaining 4.76 ms execution latency and an energy cost per inference of 0.243 mJ, showing that this solution is suitable for implementation on resource-constrained devices for real-time control.

– **Event-based low-power and low-latency estimation of hand kinematics from sEMG.** I present (5.2) the first event-based EMG encoding applied to the regression of hand kinematics suitable for working in streaming on a low-power microcontroller (STM32 F401, mounting ARM Cortex-M4). The motivation for event-based encoding is to exploit upcoming neuromorphic hardware to benefit from reduced latency and power consumption. The achieved MAE is 8.8 ± 2.3 degrees on 5 degrees of actuation on the public dataset NinaPro DB8, comparable with the SoA DNNs. The method uses 9× less memory and 13× less energy per inference, with 10× shorter latency per inference than the SoA deep net, proving suitable for resource-constrained embedded platforms.

– **Event-based estimation of hand forces from High-Density sEMG.** I present (5.3) an event-based sEMG encoding for multi-finger force estimation implemented on an MCU. This is the first work to target the HYSER High-Density (HD)-sEMG dataset in multi-day conditions closest to a real scenario without a fixed force pattern. The MAE of $(8.4 \pm 2.8)\%$ of the Maximum Voluntary Contraction (MVC) is on par with SoA works on easier settings such as within-day, single-finger, or fixed-exercise. This solution for HYSER's hardest task is deployed on a parallel ultra-low power MCU, getting an energy consumption below 6.5 uJ per sample, 2.8× to 11× more energy-efficient than SoA single-core solutions, and a latency below 280 us per sample, shorter than HYSER's HD-sEMG sampling period, thus compatible with real-time operation on embedded devices.

The contributions outlined above are declined in the following chapters as follows. Chapter 2 provides the background of the topics of the presented research: TCNs,

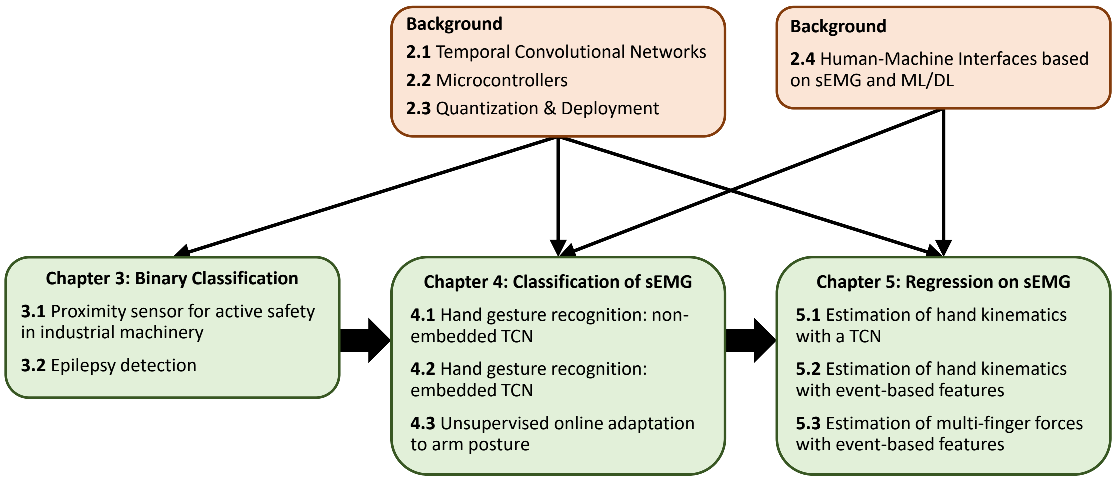
Background
2.1 Temporal Convolutional Networks
2.2 Microcontrollers
2.3 Quantization & Deployment
Background
2.4 Human-Machine Interfaces based on sEMG and ML/DL
Chapter 3: Binary Classification
3.1 Proximity sensor for active safety in industrial machinery
3.2 Epilepsy detection
Chapter 4: Classification of sEMG
4.1 Hand gesture recognition: non-embedded TCN
4.2 Hand gesture recognition: embedded TCN
4.3 Unsupervised online adaptation to arm posture
Chapter 5: Regression on sEMG
5.1 Estimation of hand kinematics with a TCN
5.2 Estimation of hand kinematics with event-based features
5.3 Estimation of multi-finger forces with event-based features

Figure 1.1: Scheme of the contents of this thesis. Chapter 2 provides the background of the three contribution chapters of the thesis, which constitute a progression as to the complexity of time-series ML/DL task: binary classification (Chapter 3), multi-class (single-label) classification (Chapter 4: sEMG-based gesture recognition), and regression (Chapter 5: sEMG-based estimation of hand kinematics and forces).

MCUs, quantization and deployment tools, and sEMG-based HMIs based on automated learning. Chapter 3 exposes the research on binary classification, addressing the task of embedded proximity sensing for industrial machinery safety and the task of epilepsy detection. Chapter 4 exposes the contribution in the topic of classification, focused on sEMG-based gesture recognition. Chapter 5 presents the research in the domain of regression, which is devoted to sEMG-based estimation of hand kinematics and forces exploiting both TCNs and event-based handcrafted features. Finally, Chapter 6 draws the conclusions of the presented research. Figure 1.1 schematizes the relationship between the background topics and the individual research contributions.

1.1.1 Publication-related structure

The core of this thesis lies in the contribution chapters, namely Chapters 3, 4, and 5, which are based on the works published during the years of the Ph.D. programme. This dissertation systematizes and harmonizes the contributions of these articles. The structure in relationship with publications is as follows:

- **Chapter 3 – Binary Classification:**

3.1: [20] M. Zanghieri, F. Indirli, A. Latella, G. M. Puglia, F. Tecce, F. Papariello, G. Urlini, L. Benini, and F. Conti, “An extreme-edge TCN-based low-latency collision-avoidance safety system for industrial machinery,” *IEEE Access*, pp. 1–1, 2024. DOI: 10.1109/ACCESS.2024.3357510

3.2: [21] M. Zanghieri, A. Burrello, S. Benatti, K. Schindler, and L. Benini, “Low-latency detection of epileptic seizures from iEEG with temporal convolutional networks on a low-power parallel MCU,” in *2021 IEEE Sensors Applications Symposium (SAS)*, 2021, pp. 1–6. DOI: 10.1109/SAS51076.2021.9530181

- **Chapter 4 – Classification: sEMG-based Hand Gesture Recognition:**

4.1: [22] M. Zanghieri, S. Benatti, F. Conti, A. Burrello, and L. Benini, “Temporal variability analysis in sEMG hand grasp recognition using temporal convolutional networks,” in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2020, pp. 228–232. DOI: 10.1109/AICAS48895.2020.9073888

4.2: [3] M. Zanghieri, S. Benatti, A. Burrello, V. J. Kartsch Morinigo, F. Conti, and L. Benini, “Robust real-time embedded EMG recognition framework using temporal convolutional networks on a multicore IoT processor,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 2, pp. 244–256, 2020. DOI: 10.1109/TBCAS.2019.2959160

4.3: [23] M. Zanghieri, M. Orlandi, E. Donati, E. Gruppioni, L. Benini, and S. Benatti, “Online unsupervised arm posture adaptation for sEMG-based gesture recognition on a parallel ultra-low-power microcontroller,” in *2023 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2023, pp. 1–5. DOI: 10.1109/BioCAS58349.2023.10388902

- **Chapter 5 – Regression: sEMG-based Estimation of Hand Kinematics and Force:**

5.1: [24] M. Zanghieri, S. Benatti, A. Burrello, V. J. Kartsch Morinigo, R. Meattini, G. Palli, C. Melchiorri, and L. Benini, “sEMG-based regression of hand kinematics with temporal convolutional networks on a low-power edge microcontroller,” in *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, 2021, pp. 1–6. DOI: 10.1109/COINS51742.2021.9524188

5.2: [25] M. Zanghieri, S. Benatti, L. Benini, and E. Donati, “Event-based low-power and low-latency regression method for hand kinematics from surface EMG,” in *2023 9th International Workshop on Advances in Sensors and Interfaces (IWASI)*, 2023, pp. 293–298. DOI: 10.1109/IWASI58316.2023.10164372

5.3: [26] M. Zanghieri, P. M. Rapa, M. Orlandi, E. Donati, L. Benini, and S. Benatti, “Event-based estimation of hand forces from high-density surface EMG on a parallel ultra-low-power microcontroller,” *IEEE Sensors Journal*, pp. 1–1, 2024. DOI: 10.1109/JSEN.2024.3359917

Chapter 2

Background

This chapter will introduce the general concepts that underlie the works presented in this dissertation. In particular, it will first explain Temporal Convolutional Networks, the category of models most used in this research work; then, it will illustrate the categories of microcontrollers of interest for the thesis. After that, this chapter will overview the techniques and frameworks employed to adapt and deploy deep models onto embedded platforms, focusing on quantization. Finally, the chapter will introduce the topic of Human-Machine Interfaces based on surface electromyography and Machine/Deep Learning, to which most of the present work is devoted.

2.1 Temporal Convolutional Networks

Temporal Convolutional Networks (TCNs) are a flavor of Convolutional Neural Networks (CNNs) based on 1-dimensional (1D) convolutions [15], [27]. Whereas classical 2-dimensional (2D) convolutions make ordinary 2D-CNN a natural choice for extracting data from digital images, 1D convolutions make TCN a natural algorithm for processing time-series data. TCNs have had successes in tasks of time-series analysis, achieving SoA results regarding both statistical accuracy and execution energy-efficiency. In particular, TCNs overperformed both ordinary non-deep machine learning models and deep Recurrent Neural Networks (RNNs), like Gated Recurrent Unit (GRU)-based models and Long Short-Term Memory (LSTM) networks. As to RNNs, TCNs are typically more easily trainable since they are less affected by the effect of vanishing or exploding gradients, and they require less training memory compared to RNNs for long input series; as to inference, TCNs have advantages in terms of data locality and arithmetic intensity, which is the general feature that makes CNNs more latency- and energy-efficient compared to RNNs.

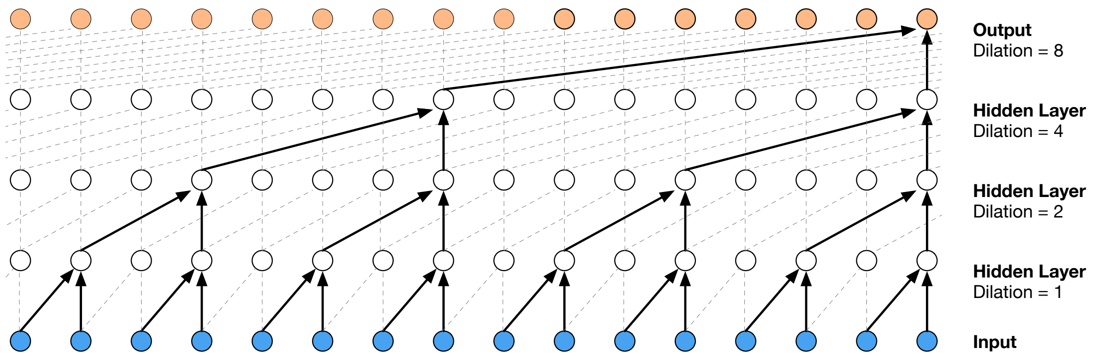

Figure 2.1: Standard representation of a deep neural network block composed of causal dilated convolutions. The filter size is $K = 3$ and the dilation factors are $d_{1,2,3,4} = 1, 2, 4, 8$. Thanks to dilation, the receptive field covers a long time window of the input. Image source: [28].

The fundamental component of a TCN is the 1D convolutional layer. The other classical layers, such as batch-normalization (BNs), non-linearities, pooling, and fully connected ending blocks, are the same as ordinary 2D-CNNs. In general, TCNs' 1D convolutional layers can be characterized by the properties of causality and dilation, which have a particular meaning for input and activations that have the nature of time sequences. **Causality** ensures that no output y_t of a convolution is computed using input elements corresponding to a future time, thus enforcing the time-consistency of the computation; so, the convolution's outputs y_t only depend on the finite set of input samples in the past $x_{t-K+1} \ldots, x_t$, weighted by the temporal convolutional kernel of length K. **Dilation** (also appliable for 2D convolutions) is the stratagem employed in TCNs for widening the convolution receptive field while keeping constant the model size and the amount of computation; a fixed interval d is inserted between the input samples taken into account by the convolution kernel. Therefore, a general temporal convolution operation characterized by both causality and dilation is expressed by the formula

$$y_t^{c_{\text{out}}} = \sum_{c_{\text{in}}}^{C_{\text{in}}-1} \sum_{k=0}^{K-1} W_k^{c_{\text{out}} c_{\text{in}}} x_{st-dk}^{c_{\text{in}}} \tag{2.1}$$

for $c_{\text{out}} = 0, \ldots, C_{\text{out}} - 1$ and $t = 0, \ldots, T - 1$; where x and y are the convolutions' input and output, respectively, t is the index of the time sample, T is the sequence length, W is the tensor of kernel parameters, c_{in} and c_{in} are the indices of the input and output channels, respectively, C_{in} and C_{in} are the total numbers of input and output channels, respectively, K is the temporal size of the filter, s is the stride (analogous to 2D convolutions), and d is the dilation. The receptive field of this convolutional layer has an extension of $F = (K - 1)d + 1$. A typical network block based on causal dilated convolutions is shown in Figure 2.1.

It is worth remarking that causality and dilation are possible TCN features, but they are by no means a mandatory prescription for effectively processing time series. As to causality, in the embedded TCN applications developed in this research, the input time window is always fully available at inference time, and inference is executed as soon as the window data are fully acquired so that considering a symmetrical neighborhood of a sample does not imply any leakage of data from the future. Regarding dilation, the emphasis on it derives from the successful heuristics in originary temporal deep models, such as the WaveNet architecure [28], which proved to benefit from a modular structure where the i-th convolutional layer of each model had dilation factor $d_i = 2^i$; for instance, in WaveNet this function was applied using blocks of 9 layers, thus with a dilation up to $d_9 = 2^9 = 512$. In contrast, in the embedded applications of this thesis, the goal is to pursue accuracy with models that are as lightweight as possible; this proved feasible with a number of convolutional layers of the order of 10, for which a specific periodic pattern for dilation did not prove beneficial. So, in the works presented here, both causality and dilations are treated as possible model properties and are selected when beneficial for accuracy. Moreover, some works on TCNs emphasize the presence of residual connections [15] in a strong relationship with ResNet [29]. However, residual connections only sometimes prove beneficial in practice; in particular, they are not used in the work presented in this thesis.

2.2 Microcontrollers of Interest

Embedded computing platforms are becoming increasingly pervasive due to their suitability for many computing tasks related to everyday life applications, such as the domains of the Internet of Things (IoT) and wearable devices. The focus of this dissertation is on Microcontroller Units (MCUs). In particular, the research and engineering interest common to all the presented works is in pursuing ML and DL solutions that keep into account the requirements for deployment and execution on MCUs. This section will illustrate the main MCU of interest for this thesis: the STM32H7 and the PULP-based GAP8 and GAP9. It is also worth noting that they mount processors that belong to two different families of Instruction Set Architectures (ISAs): ARM and RISC-V, respectively.

STM32H7 The STM32H7 microcontroller by STMicroelectronics mounts one ARM Cortex-M7 core [30]. It is designed to maximize the computational capability for general-purpose computing in a tight power envelope. Its block diagram is reported in Figure 2.2. The STM32H7 MCU features two caches to improve performance at the cost of increased energy consumption; its cache system consists of a data cache and an instruction cache,

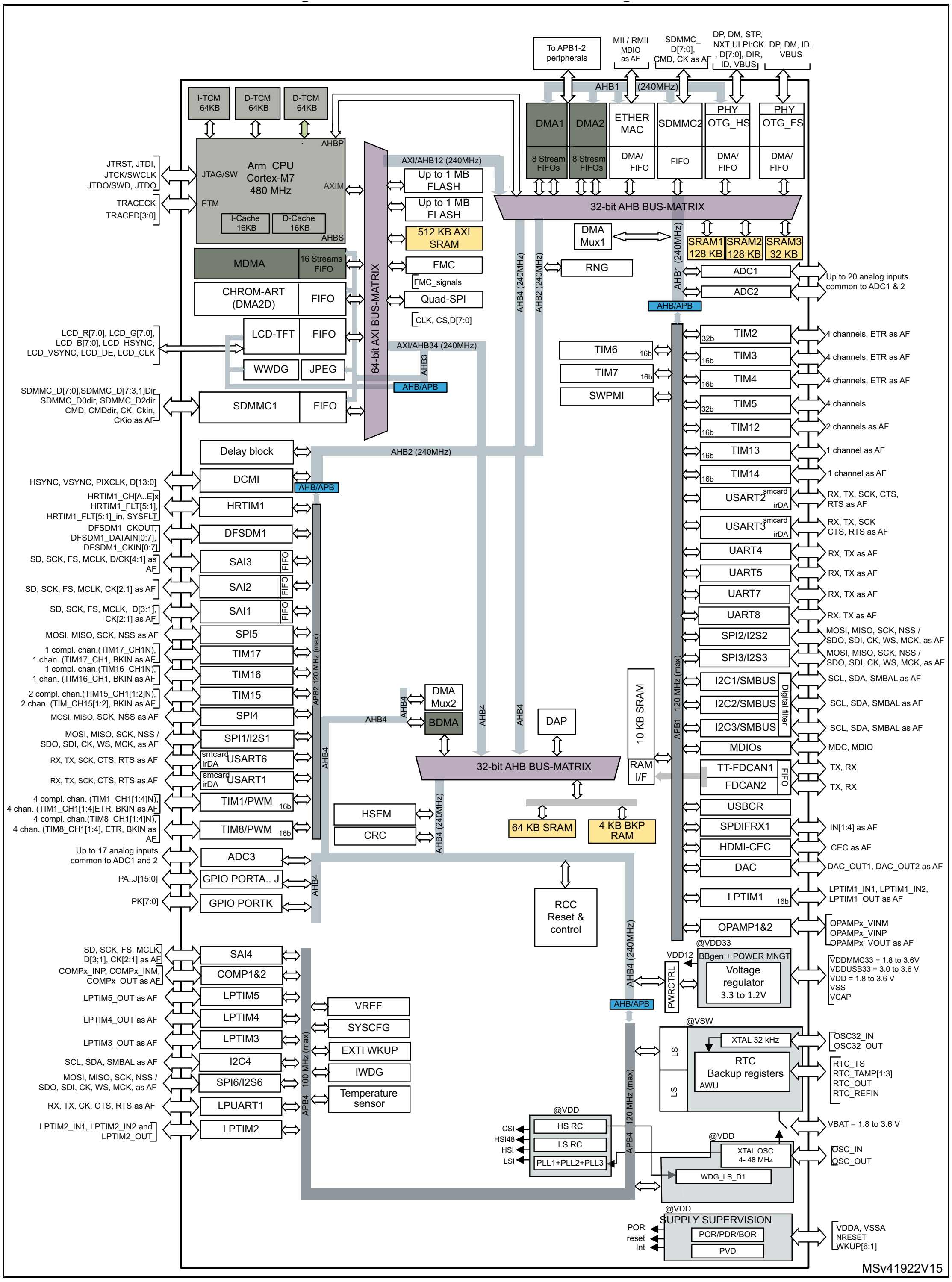

Figure 2.2: Block diagram of the STM32H7 MCU (sub-family STM32H743xI/G). Image source: [31].

thus reducing the time to both fetch instructions and load data to the register file. The ARM Cortex-M7 processor allows frequency up to 480 MHz at a power consumption of 234 mW. This MCU has several peripherals, including a Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I^2C), as well as Analog-to-Digital Converters (ADCs) for sensor data acquisition.

PULP: GWT GAP8 and GAP9 In contrast to the ARM ISA, the RISC-V ISA is open-source, allowing for several public specialized extensions (e.g., Digital Signal Processing – DSP). This freedom has allowed the development of a new architectural approach on top of the RISC-V ISA toward the design of general-purpose MCUs featuring specialized components for the acceleration of tasks such as deep learning; this advance has involved specialized accelerators (co-processors) and hierarchical memories designed to exploit the data regularity. A major example of these approaches is the Parallel Ultra-Low Power (PULP) computing, which exploits near-threshold computing targeting a high energy-efficiency and exploits parallelism to improve the performance degradation at low-voltage [32]. The PULP paradigm is based on optimizing the RISC-V ISA for DSP and DNNs, heterogeneous parallel acceleration (i.e., architecturally different compute units dedicated to unrelated tasks), and explicitly managed memory management. The extensions to the ISA include Single Instruction Multiple Data (SIMD) Multiply-and-Accumulate (MAC) operations, which are the core of DNN computation, as well as load/store instructions with post-increment, which include index updates in the memory operations. Currently, most implementations of the PULP paradigm are based on a SoA single-core MCU, termed the **Fabric Controller**, with a standard set of peripherals that offload the computation-intensive tasks to a programmable parallel accelerator, termed the **Cluster**, that features additional cores and has its own voltage and frequency domain. GreenWaves Technologies (GWT) GAP8 [33], [34], whose block diagram is reported in Figure 2.3, is a commercial PULP-based MCU featuring 9 extended RISC-V cores (one I/O plus an 8-core cluster) and is one of the most advanced MCUs specialized for DNNs. GAP8'ss8 cluster has 8 four-stage in-order single-issue pipeline RI5CY cores [36] implementing the RISC-V `RV32IMCXpulpV2` ISA. The `XpulpV2` is a domain-specific extension for efficient DSP with hardware loops, post-modified access load/store, and SIMD instructions down to 8-bit vector operands. All cluster's cores share the first level of the memory hierarchy, a 64 kB multi-banked L1 Tightly-Coupled Data Memory (TCDM) accessible through a high-bandwidth, single-cycle-latency logarithmic interconnect [36]. A cluster DMA [37] manages the data transfers between the L1 TCDM memory and a second-level L2 512 kB-memory, also managed as a scratchpad, available in the SoC domain, with a bandwidth up to 2 GB/s and a latency of 80 ns at the maximum frequency. An autonomous I/O sub-system termed the I/O DMA [38]

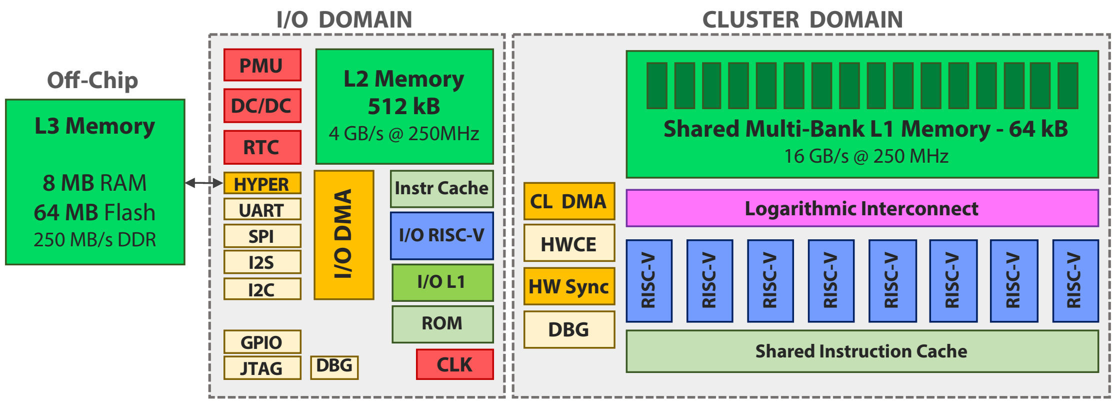

Figure 2.3: Block diagram of the GWT GAP8 microcontroller. Image source: [35].

interfaces the L2 memory with the external world and the Cypress Semiconductor's L3 HyperRAM/HyperFash memory module featured by the GAPuino board. Via the HyperBus interface, the external L3 can be connected to the system, offering an additional 64 MB storage for read-only data on Flash and 8–16 MB for volatile data on DRAM with a bandwidth up to 200 MB/s. GWT GAP9 [34] is the more recent iteration of GWT GAP8, with analogous architectural principles and architecture. GAP9 represents the SoA of low-power processors since it ranked first in latency and energy consumption on the benchmarks MLPerf Tiny v1.0 [39].

2.3 Embedding Deep Networks: Compression & Deployment

In recent years, Deep Learning has become the major solution to a vast set of computational tasks. Deep models are widely and increasingly integrated into the digital domain of industrial, consumer, and healthcare applications and products. The pervasiveness and proliferation of Internet of Things (IoT) devices, now moving toward ubiquitous cognitive computing, push the application-oriented Machine Learning to port inference to edge devices to enable real-time data processing and reduce the communication and computing load on cloud networks. However, edge platforms are strictly resource-constrained regarding memory budget and power consumption; at the same time, real-time processing of time-series data poses additional crucial constraints related to computation latency. Some advanced edge computing platforms are equipped with specialized hardware accelerators, which require hardware-specific programming efforts to be exploited efficiently. These developments and challenges have given birth to the field of Tiny Machine Learning (TinyML), the novel domain whose purpose is to enable the porting of models onto edge devices characterized by strict requirements in terms of memory and power envelope, such as embedded microcontrollers. At the algorithmic and software level of TinyML research and engineering, the set of compression techniques is crucial to reducing the resource requirements of training and inference. The methods used in the works of this thesis to reduce the amount of memory and computation are addressed to deep models' inference.

2.3.1 Quantization

In the research work reported in this dissertation, the most relevant network compression technique is quantization. Other effective methods to reduce a model's computational burden and memory footprint are pruning and vector compression. **Pruning** [40] techniques detect and remove the neurons or connections (i.e., parameters) that are redundant, i.e., do not significantly affect the network's output and are thus not relevant for accuracy. For instance, this is the case for weights and activations that are 0. Pruning can be applied at training time (*static pruning*) or at runtime (*dynamic pruning*) on either neurons or parameters. **Vector compression** techniques address the model's constants, reducing their size by clustering and sharing the weights and the biases exploiting algorithms such as K-means or hash functions. An in-depth review of deep network compression techniques is out of the scope of this thesis, and the present exposition will focus on quantization. Quantization belongs to the category of approximate computing since it approximates models' floating-point values (mathematically, real numbers) to integer values with a lower bit-width, thus allowing reduced-precision computation [40]–[42]. Typically, deep networks are trained using the `float32` or `float16` format for parameters and activations. However, smaller-bit-width representations can significantly optimize memory utilization and inference performance with a negligible loss in accuracy. In general, experimenters and model developers can configure different numerical precision for activations and parameters of each network layer, thus implementing mixed-precision models to reach the desired trade-off between model size and computation on one side and inference accuracy on the other. Quantization is more effective when applied at training time (Quantization-Aware Training - QAT); however, a popular alternative is represented by Post-Training Quantization (PTQ) methods.

In the works reported in this thesis, the considered quantization is always linear and uniform across layers. This means that all elements t_i of tensors $\mathbf{t}$ (namely: weights $\mathbf{W}$, inputs $\mathbf{x}$, hidden activations $\mathbf{a}$, and outputs $\mathbf{y}$)) having dynamic range $[\alpha_{\mathbf{t}}, \beta_{\mathbf{t}})$ are mapped to tensors of N-bit integers $\widehat{\mathbf{t}}$ via the bijective correspondence

$$t_i = \alpha_{\mathbf{t}} + \varepsilon_{\mathbf{t}} \widehat{t}_i, \qquad \text{with} \qquad \widehat{t}_i \in \mathbb{Z} \tag{2.2}$$

where

$$\varepsilon_{\mathbf{t}} \triangleq \frac{\beta_{\mathbf{t}} - \alpha_{\mathbf{t}}}{2^N - 1} \tag{2.3}$$

and the scalar $\varepsilon_{\mathbf{t}}$ is termed the *quantum* since it is the smallest possible difference between values in the quantized tensor. In general, a model's layer is composed of a sequence of three operators: a linear operation, an optional BN, and a non-linear activation function; the latter is transformed in a Quantization/Activation, in the sense that in typical procedure it is in charge of quantizing the pre-activation resulting from the linear operators [41]–[43]. With no loss of generality, we can assume $\alpha_{\mathbf{x}} = \alpha_{\mathbf{a}} = \alpha_{\mathbf{y}} = 0$ for all the inputs of linear operations and outputs of Quantization/Activations operators, but not for weights. If the original activation function is a Rectified Linear Unit (ReLU), the activations automatically satisfy the assumptions; otherwise, simple transformations can enforce the assumption. All operators can be mapped in the integer domain exploiting Equation 2.2. For linear layers:

$$\varphi = \sum_n \mathbf{W}_{mn}\mathbf{x}_n \qquad \longrightarrow \qquad \widehat{\varphi} = \sum_n \widehat{\mathbf{W}}_{mn}\widehat{\mathbf{x}}_n; \tag{2.4}$$

whereas for BNs:

$$\varphi' = \kappa \cdot \varphi + \lambda \qquad \longrightarrow \qquad \widehat{\varphi}' = \widehat{\kappa}\widehat{\varphi} + \widehat{\lambda}. \tag{2.5}$$

Furthermore, for inference, the BN parameters can be merged-pair-wise:

$$\kappa \triangleq \frac{\gamma}{\sigma}, \qquad \lambda \triangleq \beta - \mu\frac{\gamma}{\sigma}. \tag{2.6}$$

The dot-product operation in Equation 2.4 induces a shrinking of the quantum, in the sense that the quantum of the representation of $\widehat{\varphi}$ is

$$\varepsilon_{\varphi} = \varepsilon_{\mathbf{W}}\varepsilon_{\mathbf{x}} \ll \varepsilon_{\mathbf{W}}, \varepsilon_{\mathbf{x}} \tag{2.7}$$

since, typically, $\varepsilon_{\mathbf{W}}, \varepsilon_{\mathbf{x}} \ll 1$. Thus, higher precision is required to represent the integer output of the linear operator $\widehat{\varphi}$ (e.g., 32 bits), compared to its inputs and weights, before re-quantizing it upon finishing the accumulation. An analogous effect happens in the BN layers for the output $\widehat{\varphi}'$. Finally, the final Quantization/Activation operator applies the non-linearity and collapses the accumulator to a smaller bit-width:

$$\widehat{\mathbf{y}} = m\widehat{\varphi}' \mathbin{>>} d, \qquad \text{with} \qquad m \triangleq \left\lfloor \frac{\varepsilon_{\widehat{\varphi}'}}{\varepsilon_{\mathbf{y}}} 2^d \right\rfloor \tag{2.8}$$

where >> denotes the right-shift operation. The integer d is selected during the quantization procedure in such a way that $\varepsilon_{\widehat{\varphi}}/\varepsilon_{\mathbf{y}}$ can be represented with sufficient accuracy. A technique analogous to Equation 2.8 is also employed when a network has multiple branches, each with its own quantum ε, that reconverge into a single tensor, typically by a summation; so, the branches are *requantized*, i.e., brought to the same common quantum.

In the works presented in this dissertation, the deep models are quantized to a precision of 8 bit, thus producing and deploying `int8` weights and `uint8` activations; 32-bit values are used for the accumulators and the BN parameters $\widehat{\varphi}$, $\widehat{\varphi}'$, $\widehat{\kappa}$, and $\widehat{\lambda}$, as well for the quantities m and d governing the quantization and requantization procedure.

2.3.2 Frameworks and tools for embedded deep inference

This subsection provides an overview of the most relevant deep learning framework for the deployment and inference of deep networks on edge computational platforms. Nowadays, these tools are essential to enable SoA deep learning applications in both research and industry. The key feature common to the major frameworks for deep networks deployment is that they support and partially automatize some of the compression techniques described in Subsection 2.3.1, which are required for tailoring deep neural networks to edge computing platforms [44].

TFLite TensorFlow Lite (TFLite) [45] is a lightweight tool for inference on edge devices, based on the popular framework TensorFlow [46]–[48]. TFLite enables post-training quantization, also supporting the half-precision float (`float16`) and `int8` data types. In addition, TFLite is compatible with the quantization-aware training and pruning performed with the tools provided in TensorFlow or Keras [49]: the networks produced with these frameworks (situated upstream in the typical model deployment pipeline) can be imported and handles in TensorFlow Lite Micro [50], a runtime framework designed to execute deep inference on microcontrollers.

ONNX Open Neural Network Exchange (ONNX) [51], [52] is an open standard for formats of Artificial Intelligence models. In a stricter sense, ONNX is an open-source, machine-independent format for deep models. Its purpose is interoperability, i.e., simplifying the exchange of models across different frameworks and tools, also considering the target hardware, including mobile and edge devices. ONNX supports quantization to the `int8` format, both at training time and runtime (i.e., inference time) for convolutional, fully-connected, and activation layers. This support allows the execution of models in a framework different from the one used for training, enabling more flexible combinations between SoA frameworks when implementing a deep learning pipeline or product.

Apache TVM Apache Tensor Virtual Machine (Apache TVM) [53], [54] is an open compiler stack dedicated to the end-to-end compilation of deep models developed in TensorFlow (2.3.2), ONNX (2.3.2), Keras, or MXNet. The compilation is oriented to several backend frameworks and hardware target platforms. Apache TVM supports block sparsity as well as model quantization down to sub-byte precision (e.g., $1-$bit or $4-$bit). Furthermore, the microTVM [55] extension offers a C runtime that allows targeting resource-constrained bare-metal devices.

STM32 CubeAI STM32 CubeAI [56] is a software extension for the code generation tool STM32 CubeMX [57]. It provides a GUI that allows users to program STM32 microcontrollers to execute inference of deep models. STM32 CubeAI is compatible with TFlite (2.3.2) and ONNX (2.3.2) networks and can apply post-training compression. The code generated by the tool exposes APIs to implement multiple models in the same codebase and to accelerate inference execution exploiting ARM CMSIS [58] kernels.

An exhaustive review of the current compression and deployment frameworks is outside of the scope of the present thesis. Here, it suffices to mention that, in addition to the most popular SoA frameworks illustrated above, some novel specialized tools are available to implement more advanced compression techniques or quantization in a finer-grained fashion, such as QKeras [59], Larq [60], and Brevitas [61]. For the scope of this thesis, it is more relevant to illustrate the quantization and deployment tools developed at the Energy-Efficient Embedded Systems Laboratory of University of Bologna and the Integrated Systems Laboratory of ETH Zürich within the Parallel Ultra-Low Power (PULP) Platform project [62], [63], that were extensively used in the work, namely NeMO, QuantLib and DORy.

NeMO NEural Minimizer for pytOrch (NeMO) [43], [64] is an open-source Python library for quantizing neural networks implemented in PyTorch [65], [66]. NeMO is oriented at the deployment onto highly memory-constrained, ultra-low power computation devices, with a particular focus on PULP-based microcontrollers [63]. NeMO implements the quantization technique of PArameterized Clipping acTivation (PACT) [67] as well as other methods, and it allows the configuration of the quantization bit-width of activation, weights and BN parameters, as well as BN folding. NeMO's quantization can be mixed-precision, and the tool also offers a semi-automatized precision relaxation. NeMO's pipeline of model transformation toward quantization works on three levels of the network's representation, `torch.nn.Module` [68] and `torch.autograd.Function` [69]. The sequence of transformations starts from the original floating-point model, termed full-precision (`FP`) stage, applying a first quantization and calibration that yields the Fake-Quantized (`FQ`) stage, where the model is still trainable; then, the quantization is frozen yielding the Quantized Deployable (`QD`) stage, which has discretized floating-point values; finally, proper *integerization* yields the Integerized Deployable (`ID`) stage model. The `ID` model is the final quantized format to be exported to lower-level deployment tools: this model stage is bit-accurate, i.e., contains the actual values to be deployed, and NeMO implements export to ONNX 2.3.2 format. The illustrated pipeline allows NeMO users to perform both Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). The pipeline

$$\texttt{FP} \rightarrow \texttt{FQ} \rightarrow \text{calibration} \rightarrow \texttt{QD} \rightarrow \texttt{ID} \tag{2.9}$$

(possibly including BN folding) is a Post-Training Quantization (PTQ). However, it is worth noting that the `FQ`-stage model is not frozen and still allows fine-tuning via backward gradients.

QuantLib Quantization Library (QuantLib) [70] is an open-source quantization library that works with the same purpose as NeMO (2.3.2) and analogous internal mechanics; however, QuantLib is more recent and currently more actively maintained than NeMO. QuantLib is also a component of Quantization Laboratory (QuantLab) [71], [72], along with organizing software to manage larger-scale machine learning, e.g., by enabling multi-GPU acceleration of net training with `torch.nn.DataParallel` exploiting Horovod[73]. Superseding NeMO (2.3.2), QuantLib/QuantLab are currently the go-to quantization tools within the PULP Platform project, with applications in image recognition [74], epilepsy detection [75], [76], and neuromorphic obstacle avoidance by uncrewed aerial vehicles [77]. In the works reported in this thesis, QuantLib was used alone without recourse to QuantLab.

DORy Deployment ORiented to memorY (DORy) [41], [42], [78] is an automated tool for deploying deep models onto resource-limited embedded platforms, typically with a memory budget $\leq$ 1 MB of on-chip SRAM. DORy addresses the memory constraints dealing with tiling as a Constraint Programming (CP) problem, with the strategy of maximizing the L1 memory usage under the topological constraints of each model layer. The tool generates C code to orchestrate off- and on-chip transfers and computation phases; Figure 2.4 displays an example of layer tiling. As input, DORy can receive models in the ONNX format (2.3.2) produced in NeMO (2.3.2) or QuantLib (2.3.2). DORy specializes in feed-forward neural networks with single-wire residual connections. The tool can implement layers using mixed-precision of 2, 4, 8 bits. DORy is compatible with the GWT Virtual System-on-Chip (GVSoC) [79], the simulator of RISC-V processors

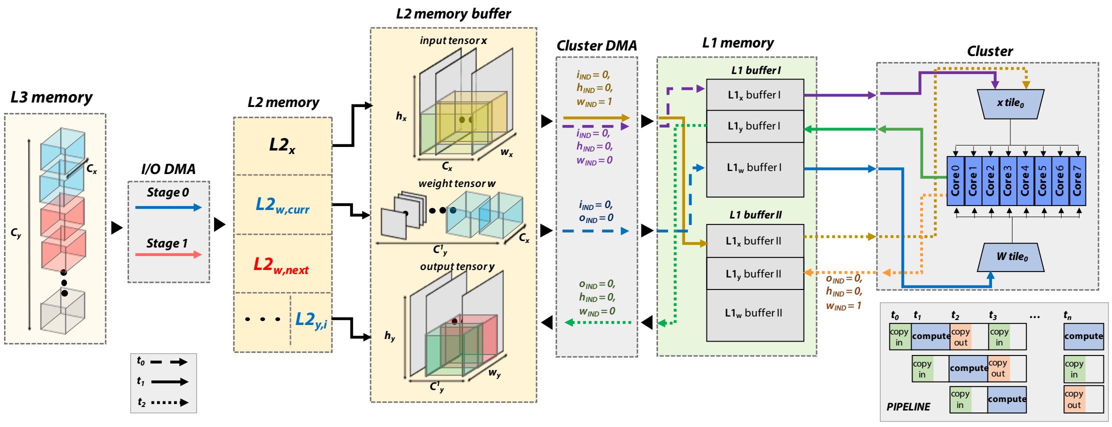

Figure 2.4: An example of a DORy routine tiling a layer across L3, L2, and L1 memories. On the left, the I/O DMA copies weight tiles in case only C_y is L3-tiled. Two buffers are used for L2$_w$. Then, the Cluster DMA manages L2-L1 communication using double-buffering, while the cores compute a kernel on the current tile stored in one of the L1 buffers. Image source: [42].

available for PULP-based platforms such as GreenWaves Technologies' GAP8 [33], [34] and GAP9.

2.4 sEMG-based Human-Machine Interfaces

Decoding hand gestures is an established method for developing advanced Human-Machine Interfaces (HMIs), which leads to a wide range of application scenarios, such as industrial robot control, gaming interfaces, prosthetic control, or augmented reality [80]–[84]. In the HMI field, movement modeling relies on processing information coming from video cameras [85] or muscular activity [86]. Camera-based techniques rely on image processing algorithms that recognize users' hands in a scene and recognize different gestures using computer vision. Although this approach can decode several different gestures reliably, it suffers from line-of-sight issues and scene illumination variability and requires pre-installed environmental cameras. On the other hand, approaches based on muscular signal analysis are inspired by the prosthetics domain, where electromyographic signals are used to control artificial hands [87], [88]. Commercial prosthetic controllers are simple and highly reliable. However, they provide a non-natural interface, unsuitable for intuitive gesture interface design because of the high level of concentration required by the user and the long learning curve.

New ML/DL approaches have been extensively explored to enable the design of natural gesture interfaces. They aim to map muscular contraction patterns onto a set of intended gestures [89], using supervised learning methods such as Support Vector Machines (SVMs), Random Forests (RFs), Linear Dscriminant Analysis (LDA) or artificial neural networks (ANNs) [90]–[92]. This section will expose the background concerning the surface electromyographic signal, the ML/DL approaches to hand gesture recognition, and the recent progress in ML/DL applied to sEMG regression to estimate hand kinematics and multi-finger forces.

2.4.1 The sEMG signal

The electromyographic (EMG) signal [93]–[95] is the bioelectric potential originating from the current generated by the ionic flow through the membrane of the muscular fibers, and it is, therefore, a major index of the muscular activity. This potential is generated by the electrical stimulus starting from the central nervous system and passing through the motor neurons (motoneurons) that innervate the muscular tissue. Typically, the EMG signal has amplitude ranging from $10\,\mu\text{V}$ to $10\,\text{mV}$, and bandwidth $\sim 2\,\text{kHz}$. Moreover, the sEMG is a very challenging signal as it is affected by several noise sources, such as motion artifacts, floating ground noise, crosstalk, and power line interference [96].

EMG data can be acquired either with invasive or non-invasive methods. In the part of this thesis dedicated to the topic of sEMG ML/DL, I focus my research on surface electromyography (sEMG), the non-invasive technique that senses the EMG activity via electrodes positioned on the skin's surface. In the sEMG setup, the action potentials (APs) can be detected using an instrumentation amplifier with the positive and negative terminals connected to two metal plates positioned on the skin surface; the sEMG signal results from the superposition of all the detected APs underlying the amplifier [92]. In the field of HMIs, building gesture recognition upon the analysis of sEMG signals is one of the most promising approaches since non-invasiveness is an essential requirement for many application scenarios.

2.4.2 Classification of sEMG: hand gesture recognition

In recent years, several sEMG-based hand recognition approaches have been presented in academia and commercial applications. All of them share a typical structure, based on *i)* an analog front end for bio-potential acquisition, *ii)* a data preprocessing and feature extraction/selection step, and *iii)* a final classification back-end. Moreover, they usually all rely on ML algorithms such as SVMs, RFs, LDA, or ANNs [90]–[92], [97]–[102].

For instance, in [103], [104], the authors presented a 4 hand gesture classification with accuracy above 90%, using an ANN with 5 time-domain features (Mean Absolute Value (MAV), Mean Absolute Value Slope (MAVS), number of Slope Sign Changes (SSC), number of zero crossings (ZC), and Waveform Length (WL)). Castellini *et al.* [105] illustrated a three grasp recognition, achieving 97.1% classification accuracy using the Root Mean Square (RMS) as features extraction for an SVM. On a more general scenario (up to 50 different hand gestures), remarkable results were obtained by Atzori *et al.* [91] on the Non-Invasive Adaptive hand Prosthetics Databases 1, 2, and 3 (NinaPro DB1, DB2, and DB3), employing a mixture of time- and frequency-domain features. As a downside, all these works are limited to a single-session setup. This setup fails to tackle the issue of the inter-session accuracy drop observed when classifying gestures from a never-seen session after training on just one session.

As a result, the crucial challenge in sEMG-based gesture recognition has shifted from absolute classification accuracy to managing the variability of the signal, which is affected by several factors such as anatomical variability, posture, fatigue, perspiration, changes in the skin-to-electrode interface, user adaptation, and electrode repositioning over multi-day usage [90], [92], [106], [107]. These factors strongly hamper generalization, thus limiting the long-term use and the realization of robust real-time recognition systems. For instance, Benatti *et al.* [108] and [107] collected sEMG data from several subjects in multi-day campaigns to analyze the performance degradation of conventional ML algorithms when donning and doffing the sensory setup. In these experiments, the inter-session accuracy drop after training on a single session was up to 30%. The proposed solutions mostly rely on extending the training datasets, modifying the acquisition setup (e.g., increasing the electrode count), and extracting a broader set of features to improve algorithm convergence. These solutions lower the average accuracy drop, decreasing the average error rate to 12% [90]. However, this performance drop and the lack of generalization are still hampering the deployment of these solutions in reliable, commercially available systems.

A new state-of-the-art strategy to robustify recognition against temporal variability is multi-session training. This strategy has been made possible by the release of multi-session sEMG datasets such as the Non-Invasive Adaptive hand Prosthetics Database 6 (NinaPro DB6, 10 sessions, 8 classes) [107] and the University of Bologna - INAIL (UniBo-INAIL) database (8 days $\times$ 4 arm postures, 6 gestures) [92]. On the NinaPro DB6, Palermo *et al.* [107] reached an inter-session accuracy of 25.4% by feeding Wave Length to a Random Forest. Cene *et al.* [109] successfully employed Extreme Learning Machines (ELMs) to raise this inter-session accuracy to 41.8%. It is worth noticing that the reason why the accuracy reached on the NinaPro DB6 is much lower than the one reached on other datasets with a similar number of classes and sensors is that the hand movements of NinaPro DB6 are all grasp gestures, thus much less diverse and discernable than the gestures in ordinary datasets. On the Unibo-INAIL dataset, Milosevic *et al.* [92] showed that multi-posture and multi-day training improve inter-session generalization. A Radial Basis Function kernel SVM (RBF-SVM) applied on 4-channel single samples of the RMS signal yielded an intra-session recognition accuracy higher than 90%, with an inter-session accuracy drop up to 20% (a value similar to [107], [109]). The aforementioned approaches showed the major limitation of classical ML: it strongly relies on domain-specific knowledge and hand-crafted features, limiting the capability to generalize over time.

To cope with this issue, DL represents a valid approach since it incorporates feature learning into model training and can reach a better generalization on the data. DL-based solutions have also been prompted by increased data availability (public sEMG benchmark databases) and significant improvements in computing hardware [110]. Table 2.1 shows that DL methods outperform traditional ML approaches when classifying data from different sessions. This conclusion is also reinforced in the survey conducted by Phinyomark *et al.* [106]. The first end-to-end DL architecture was proposed by Park and Lee [111], who applied a CNN + RMS on NinaPro DB1, outperforming an SVM in classification accuracy across subjects. From our variability point of view, it is interesting to note that this early work already addresses inter-subject variability, showing that a CNN benefits more than an SVM from an adaptation phase introduced before classifying data from unseen subjects. Atzori *et al.* [115] also proposed a CNN-based approach to recognize the 52 hand gestures from the NinaPro DB1, DB2, and DB3 (taking 150 ms-windows of RMS, acting on time$\times$channels), reaching classification accuracy comparable to classical methods such as RF.

Regarding the issue of variability, a strategy typical of DL is Adaptive Batch-Normalization (AdaBN) [116], a domain adaptation consisting in re-training the BN layers [117] of deep models without fine-tuning the entire network. AdaBN is parameter-free, free of additional components, and computationally simpler than generalized fine-tuning. These qualities make AdaBN interesting for real-time setup, in which it has already shown some success. For instance, Du *et al.* [114] employed a CNN + instantaneous High-Density (HD) sEMG images, attaining 63.3% accuracy on the 8 classes of their CapgMyo database.

Recently, TCN approaches have started appearing in recent research, gaining traction for sEMG-based gesture recognition. Tsinganos *et al.* [112] achieved 89.8% classification accuracy on the 53 classes of NinaPro DB1 with an RMS-fed TCN. This result is 4.8% better than SoA [118] and surpasses by 19.3% the previous results from the same authors obtained with conventional 2D-CNNs [99]. This TCN was evaluated using a receptive field (i.e., input sequence lengths) of 300 ms up to 2.5 s. Although the NinaPro DB1 dataset is not multi-session [91] and so does not involve the temporal

Table 2.1: Comparison between SoA embedded platforms for EMG processing.

WORK	Dataset	Subjs	Sessions	Classes	Channels	Window	Features	Algorithm	Accuracy (%) intra/inter	Real-time? / Embedded?
Hudgins [103]	private	18	1	4	1	200 ms	MAV, ZC, SSC, WL	shallow ANN	88.9 / N.A.	no / no
Park [111]	NinaPro DB1	27	1	6 [1]	8	2000 ms	RMS time×ch.	CNN	N.A./∼94 [2] [3]	no / no
Tsinganos [99]	NinaPro DB1	27	1	53	8	200 ms	RMS time×ch.	CNN	70.5 / N.A.	no / no
Tsinganos [112]	NinaPro DB1	27	1	53	8	300 ms, 1200 ms	RMS	TCN	89.8 / N.A.	no / no
Betthauser [113]	private	9	1	27	8	1675 ms	200 ms-MAV	TCN	69.5 / N.A.	no / no
Hu [98]	NinaPro DB1	27	1	53	8	200 ms	RMS	CNN + LSTM	87.0 / N.A.	**yes** / no
Kaufmann [90]	private	1	121	10	8	150 ms	MAV, ZC, SSC, WL	SVM	N.A. / 87.7	no / no
Milosevic [92]	Unibo-INAIL	7	8	6	4	1 sample	RMS	SVM	∼90 [4] /∼70 [4]	no / no
Du [114]	CapgMyo	8	2	8	128	1 sample	inst. HD-sEMG images	CNN	98.6 / 63.3 [3]	**yes** / no
Palermo [107]	NinaPro DB6	10	10	8	14	200 ms	WL	RF	52.4 / 25.4	no / no
Cene [109]	NinaPro DB6	10	10	8	14	200 ms	MAV, VAR, RMS	ELM	69.8 / 41.8	no / no
Zanghieri (4.2) [3]	NinaPro DB6	10	10	8	14	150 ms	raw sEMG	TCN	54.5[5]/ 49.6	**yes / yes**
	20-session	3	20	9	8	150 ms	raw sEMG	TCN	97.1 / 93.7	**yes / yes**

[1] Restricted to 6 functional movements, without rest class.
[2] Inter-subject.
[3] With domain adaptation.
[4] Precise values depending on session and training strategy.
[5] Our lower-than-SoA intra-session accuracy is due to the fact that [109] (1) relies on single-session training, prone to overfit to the sessions; and (2) uses a very aggressive signal filtering over 200 ms time windows, jointly with outlier removal embedded in the algorithm, so as to smooth the transients or even discard them from the accuracy.

variability, [112] is a valuable demonstration that TCNs can yield good accuracy on this task. Betthauser *et al.* [113] proved that TCNs outperform Long Short-Term Memory (LSTM) networks in the sEMG-based gesture recognition task, reaching 69.5% accuracy on 27 classes. Also, the TCN used in this work has a very wide receptive field: the 1.7 s input windows were generated by computing the MAV from 200 ms-long sequences.

Overall, these works proposing TCNs for sEMG-based gesture recognition share the limitation of using very long (i.e., $\geq$ 300 ms) signal windows. In particular, in [112], the dilated convolution is used to hugely enlarge the receptive field at constant network size instead of exploiting dilation to work with smaller networks at a constant receptive field. This limitation implies two (related) issues: *i)* the comparison is altered with the other works that comply with the consensus of using time windows $<$ 300 ms [103]; *ii)* the proposed TCNs are evaluated under conditions which are not feasible for a usable real-time implementation. In contrast, in the sEMG recognition research presented in this dissertation, I focus on real-time classification and target full compliance with the upper limit of 300 ms. For example, TEMPONet, the most important TCN presente in this thesis, uses 150 ms signal windows as input and needs $<$ 15 ms for inference when deployed on an embedded platform.

2.4.3 Regression on sEMG: hand kinematics and finger force

As explained in Subsection 2.4.2, conventional ML algorithms, such as LDA, SVM, and ANN, achieve above 90% classification accuracy [92], [119] on sEMG-based hand gesture recognition, even though they are outperformed by DL models, which can leverage larger datasets, exploit feature learning, and handle time-windowing of the sEMG signal with no need for preliminary feature extraction [3], [22]. Although the use of the sEMG signal allows the conventional ML/DL approaches to control both gestures and grasps with a relative naturalness, the aforementioned systems are restricted to limited sets of predefined static positions (e.g., closed hand, pinch grasp, pointing index, etc.), which do not allow an entirely natural and versatile control. As a result, targeting hand kinematics and force with sEMG-based regression is a promising research direction, and several DL approaches could tackle regression problems.

However, most of the DL methods that are accurate for regression are computationally intensive and require a high memory footprint due to large model sizes [98], [112], [113], hampering deployability on edge devices with real-time execution constraints. Therefore, a DL reliable framework for energy-efficient platforms requires a careful multimodal hardware-software co-design. In particular, after finding a high-accuracy deep model, it is necessary to minimize its size (e.g., via quantization [67]), then determine the latency and power consumption of the model inference when it runs on the targeted embedded device. It is noteworthy that most convolutional networks proposed for sEMG reach SoA accuracy only at the cost of model size up to tens of MB [98], lacking deployment-oriented optimizations, such as stride and dilation, quantization [67], or pruning, which can cut down the model size by more than 10× with only a marginal loss in accuracy [22].

The sEMG regression task has been addressed in several inspiring works, distinguished by their target variable: kinematic (joint angle, joint velocity) or dynamic (finger force). The position of the joint angles represents the best indicator for hand kinematics, and it is therefore the most used parameter in hand gesture regression. For instance, in [120], the regression is targeted toward the joint angles of a dataglove, reaching a median R^2 of 0.63 using a Wiener filter. However, this work does not test other algorithms to improve the regression score since it is more focused on the control quality as perceived by users. In [121], an LSTM deep network is applied on the same dataset, yielding a Mean Absolute Error (MAE) of 7.04°. The limitation of this work is that it does not explore purely convolutional networks, which are more amenable to parallelization and, hence, more suitable for embedded control. Alternative approaches to hand kinematics focus on the velocity of hand joint movements. For instance, velocity was targeted in [122], adopting a hybrid classification-regression setup that thresholds speed into 3 levels, thus still limiting the prediction to discrete classes. A completely orthogonal approach for sEMG regression focuses on hand dynamics instead of kinematics. The work [123] targeted a multiple-Degrees-of-Freedom (multi-DoF) force estimation However, force estimation is a more restricted application since it is limited to grasp movements.

A major shortcoming of all the aforementioned works is that none of them addresses the problem of deployment onto embedded control devices. In particular, they do not discuss how their models cope with resource-constrained platforms nor explore techniques such as model search, quantization, and pruning.

In this thesis, I present research addressing the challenge of sEMG-based regression to decode hand kinematics and force. In particular, the work exploits a TCN to estimate hand joint angles and explores event-based feature extraction to estimate both hand joint angles and multi-finger forces. Both these use cases of time-series modeling are profiled and shown to be effective for low-latency operation on resource-constrained devices.

Chapter 3

Binary Classification

This chapter presents the novel contributions of this thesis that belong to the category of binary classification. These works are a proximity sensor for active safety in industrial machinery (3.1), which is accurate and robust against acoustic noise, and a setup for epilepsy detection from iEEG signals (3.2). The two contributions belong to two different domains: the first work is a task of applied research oriented to an industrial problem, whereas the second work is related to the clinical field. The methodology that connects these two works is the algorithmic research oriented to TCN deployed on embedded MCUs. The first two contributions in binary classification presented in this chapter demonstrate the power and versatility of the approach. In Chapters 4 and 5, the edge-oriented approach to time-series ML and DL will be applied to the progressively complex tasks of multi-class classification and multi-target regression, respectively.

3.1 An Extreme-Edge TCN-based Low-Latency Collision-Avoidance Safety System for Industrial Machinery

3.1.1 Overview

Nowadays, several industrial sectors employ autonomous moving machinery that can constitute a source of hazard and must therefore be operated by workers with specific training and skills, with well-defined safety practices and working conditions. A major concern is safeguarding operators' health. Solutions to do so with a reduced impact on the workflow of the machinery aiming at achieving high productivity and safety are currently an active field of research & development.

Industrial machines can be equipped with sensors that enable them to continuously monitor their surroundings in an automated way. This enables safeguards that halt operations and drive the machinery to a safe state if people or dangerous obstacles are detected. The technical challenge in this scenario is to make the detection robust against variations in environmental conditions across multiple deployment sites (i.e., in "space") and across several operational conditions of the same site (i.e., in "time"). Safety systems operating in an automated way belong to the domain of *functional safety* [124], where protection is framed and implemented as an *active*, *input-output* system. The safety function is the action generated in response to the processed input. Functional safety does not include *passive* systems (e.g., thermal insulation or fire-resistant doors) but involves electronics, software, and actuators.

Recently, methods based on ML, and specifically DL, have been gaining adoption in domains such as machine vision and data analytics [125]. DNNs can now be regarded as a mature methodology in data analysis. Hence, DNNs are also promising for information processing tasks of active systems for functional safety since they can integrate multiple data streams and extract information from them. As to execution, ML/DL algorithms can be run in the cloud or at the edge, i.e., locally on a platform closely connected to the devices acquiring data [126], [127]. More specifically, recent advances in the field of Tiny ML (2.3) [128]–[130] are enabling the porting of real-time ML inference onto embedded computing platforms with strict constraints in terms of memory or power envelope, such as MCUs [131], sometimes equipped with accelerators for ML/DL. For safety-critical systems, processing the data near the sensors can enhance reliability and ultra-low latency and increase the trust in ML-based solutions in industries such as manufacturing, mobility, and robotics [132], [133].

Safety solutions relying on ML/DL make it necessary to elaborate and advance the international standards that regulate functional safety. The major challenge is that the current versions of international standards do not cover novel, most recently introduced technologies and paradigms. This is an issue since innovative methods or algorithms can not be certified by definition. Hence, innovative solutions, even if proven effective for operators' safety and production efficiency, undergo a large delay before inclusion into a new version of a standard; in turn, inclusion happens when a solution is mature and able to induce industrial interest in its inclusion. For this reason, the adoption of ML/DL-based solutions in Electro-Sensitive Protective Equipment (ESPE) systems [134] has not been addressed yet by any industrial safety standard. The stance of the project's team in this regard is that interest from the industry must be fostered by showcasing innovative prototypes able to demonstrate the power of ML/DL for functional safety: this is the direction of the research presented in this work.

This section targets the specific domain of industrial woodworking machinery. It proposes a functional safety prototype for collision avoidance based on ultrasound (US) sensing and processing based on a TCN (2.1), a DNN specialized for time series. The system is able to detect persons or obstacles in the field of view of the US sensors, which are mounted on the woodworking machine in such a way as to probe the space of operation of the machine's moving parts. A detection triggers a stop of machine movement in real time. In detail, the contribution is multiple:

- This section implements a system based on 9 US sensors, an FPGA, and an MCU, mounted onto an industrial woodworking machine.
- The setup is used to collect a dataset for the detection task (i.e., clear space vs. human or obstacle), representative also of the acoustic noise conditions typical of an industrial facility, which are challenging since they impact the US signals; this curated dataset contains a total of 5085 US signal windows organized in 170 runs of the system in different obstacle and noise conditions.
- A TCN trained and tested for the purpose achieved sensitivity 96.7%, specificity 99.1%, and AUROC 0.993 in the absence of acoustic noise.
- In the presence of noise, exploiting an incremental learning technique proved that the proposed setup and model are able to leverage increasing amounts of data, attaining sensitivity 90.5%, specificity 95.2%, and AUROC 0.972.
- Deployment of the proposed TCN on the STM32H743ZI MCU yielded a profiling which outperforms the SoA TCN model for the task [135]: memory footprint of 560 B ($3\times$ smaller than SoA), with a latency of 5.0 ms and energy consumption of 8.2 mJ per inference (both $2.3\times$ less than SoA).

The proposed solution improves detection robustness against acoustic interference characteristic of a manufacturing environment, working with a resource budget fit for real-time execution on resource-constrained edge computing platforms. Table 3.1 reports a scheme of the advances of this work compared to the SoA represented by Conti *et al.* [135]. The proposed paradigm is generalizable to different sectors; in particular, the limited hardware requirements allow the scale-up of the approach, enabling adoption in scenarios with more sensors and, thus, wider monitored space in terms of the number of machines and extent of the probed areas.

For reproducibility and advance in the research & development community, I also released open-source the code developed for this research.[1] As a research group, we also released the curated dataset realized in this work.[2]

3.1.2 Related Work

3.1.2.1 Safety systems in industrial woodworking machinery

Industrial woodworking machines typically have a static base and a moving cabinet that slides horizontally at speed up to 1 m/s and operates over a working surface of the order of $4\,\mathrm{m} \times 1.5\,\mathrm{m}$ [136]. Overall, these machines have a length of $5 - 10\,\mathrm{m}$, a width around 5 m, and a height of $1 - 3\,\mathrm{m}$ [137], [138]. The moving cabinet can hit operators or objects, causing severe injuries or damage. In general, existing machine models rely on both active and non-active safety systems [136]–[140]. Non-active safety includes simple elements such as enclosures of the working units by fences, lateral curtain guards, transparent hatches, or perspex windows, based on the desired tradeoff of protection vs. accessibility and visibility. LEDs signal the machine status in real-time with a simple color code. This work focuses on more advanced active safety systems.

Active safety is based on real-time anti-collision systems required to operate while machines work at medium or maximum speed in a premise shared with workers performing regular work in the surroundings. Since detecting hazardous situations forces the machine to a safe-state mode, which can be unlocked only manually, erroneous automatic detection can cause a slowdown in the workflow. Active safety systems include: soft bumpers that stop the machine in case of accidental contact with persons or objects; pressure-sensitive floor mats; photocell barriers that detect the approach of persons or objects, automatically reduce the speed of the machine, and restore the maximum speed when the obstacle leaves the area; laser scanners that only enable the machine to start after the operator has left the area; automatic verification of the locking systems' positioning.

The proposed setup exploits US signals, processed for detecting objects or people within the space of operation of the machine. Compared to existing solutions, the proposed setup has several advantages. First, the proposed solution is a proximity sensor designed to trigger the stop of the machine before a collision, in contrast to bumpers. As to established collision avoidance systems, the existing laser scanners only probe a horizontal plane (at a height $< 1\,\mathrm{m}$ above the floor) [137], whereas the proposed ultrasound sensors probe a 3D field of view. Compared to all alternative setups, including

[1] https://github.com/MarcelloZanghieri2/edge_tcn_collision_avoidance
[2] https://github.com/MarcelloZanghieri2/collision_avoidance_ultrasound_dataset

Table 3.1: Contribution of this work in terms of the advances compared to the SoA represented by Conti *et al.* [135].

WORK	**# of US sensors**	**Realized dataset**		**Method for accuracy on noisy data**	**Embedded system**		**Reproducibility**	
		novel?	**size**		**deployed @ edge?**	**real-time?**	**published dataset?**	**published code?**
Conti *et al.* [135]	1	✓	227	data augmentation up to 1000×	✓	✓	✗	✗
This work	9	✓	5085 (22.4×)	incremental learning (augmentation just 64×)	✓ 0.32× parameter memory < 0.43× energy	✓ < 0.43× latency	✓	✓

photocell barriers, the proposed solution can improve its detection accuracy during its lifetime since the proposed DNN benefits from incremental learning from data acquired in new conditions.

It is worth remarking that the technical documentation uses the term *collision avoidance* also for potential collisions between machinery's equipment or between tools and material, handled during the virtual prototyping of the piece and the simulation and scheduling of numerical control positioning [136], [138], [139]; this kind of internal collision is not related to the topic of this work. It is also worth stressing that this work does not deal with inner systems for safety or maintenance such as air conditioning of electrical components or automatic lubrication.

A relevant earlier work tackling US-and-DL-based functional safety for woodworking machinery is by Conti *et al.* [135], who employ TEMPONet, a TCN previously applied to embedded biosignal processing in real-time [3], [24]. The TEMPONet TCN architecture is the subject of Sections 4.2 and 5.1. The previous work by Conti *et al.* [135] stemmed from the same project as the present contribution but only has the nature of a technical report documenting an incomplete stage of the research. Although a direct accuracy comparison is not viable since [135] relies on a different 1-channel dataset, it is possible to highlight several advancements (also reported in Table 3.1): (*i*) the proposed system mounts 9 ultrasound sensors, whereas the previous work mounted just 1; (*ii*) this work releases the dataset open-source; (*iii*) this work employs a smaller DNN, reducing the hardware resources and latency budget for execution; (*iv*) this work tackles a noisy environment by implementing an incremental training protocol instead of brute-force data augmentation.

3.1.2.2 Rationale of this work in relation to the established functional safety standards

All safety equipment applied on industrial machines must get certified according to standards, such as the ones by the International Electrotechnical Commission (IEC) (covering electrical, electronic, and related technologies), that define the Safety Integrity Level to be met. Machinery-halting safety systems such as [135] and the one presented in this work fall under the regulations concerning non-contact Electro-Sensitive Protective Equipment (ESPE) sensors (e.g., photodiodes). More in detail, IEC 61508 [124] regards any electrical/electronic/programmable electronic (E/E/PE) for functional safety systems, such as sensors, control logic, or actuators, and also microprocessors; EN IEC 61496 [134] focuses on the requirements of design, building, and verification of systems based on non-contact ESPEs to detect persons in a safety system, focusing on indoor environments; EN IEC 62046 [141] addresses ESPEs for human detection for safety, focusing on industrial environments with machinery.

Novel ML/DL-based research & development prototypes such as the one presented in this work are not covered by current standards, nor can they receive certification in the short term. This limitation means that, as of today, developing finalized products based on the presented proof-of-concept is not possible. *The purpose of this work is to push research and technical expertise ahead of current standards and certifications.* The motivation in undertaking the present research is to showcase how promising data-driven safety systems are, intending to incentivize both technical exploration and regulatory interest. This line of research, in addition to improving the SoA (3.1.2.1) as to hardware-software figures of merit, will stimulate the attention from the industry for this class of approaches and methods, prompting a push for the inclusion of ML/DL-based functional safety into the future versions of the standards.

3.1.3 Materials & Methods

3.1.3.1 Targeted woodworking machine

The specific industrial woodworking machine used in this work is an SCM Morbidelli X200 [142], depicted in Figure 3.1. This machine features two panels with a 3×2 and a 1×3 array of US sensors, as shown in Figure 3.2. Figure 3.3 schematizes the spatial configuration of the proposed proximity sensing system. This setup was used to collect data on the machine and test the accuracy and performance of the proposed solution. This setup is easily generalizable to machines and environments in different industrial sectors.

3.1.3.2 System architecture

The hardware architecture of the proposed system is shown in Figure 3.4 and relies on transducers that emit US pulses and sense the echo if a pulse hits an obstacle; if a detection happens, the system outputs a stop signal to the machine control. The main elements of the system are (*i*) the US sensors and their drivers, (*ii*) a Lattice FPGA for low-latency data collection, and (*iii*) a Nucleo-144 board mounting an STM32H743ZI MCU. This system performs both data collection and obstacle detection.

The data are acquired by a 2×3 plus 1×3 configuration of 9 Multicomp Pro MCUSD14A58S9RS-30C ultrasonic ceramic transducers.[3] Using 9 sensors instead of

[3]https://octopart.com/mcusd14a58s9rs-30c-multicomp-30988352

Figure 3.1: The SCM Morbidelli X200 industrial woodworking machine used in this work. Image source: SCM Group [143].

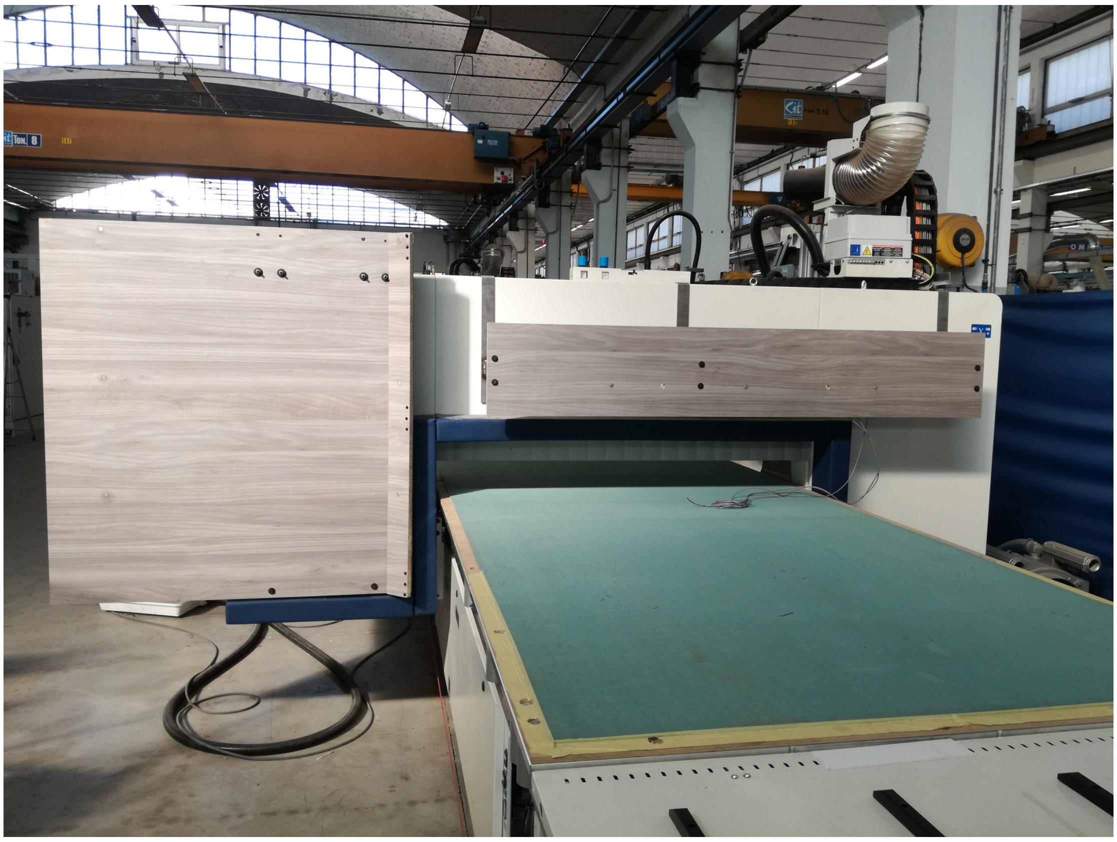

Figure 3.2: Configuration of the 9 US sensors mounted on the machine. The sensors are the grey metal round elements on the panels; the circular black pieces are washers for fastening the panels. Compare with Figure 3.3.

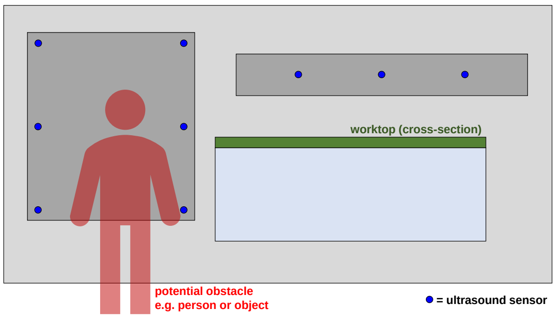

Figure 3.3: Spatial organization of the proposed proximity sensing system. The 3 US sensors on the moving cabinet over the worktop proved useful in preliminary tests to better sense the space surrounding the working table and obstacles at the far end of the working table. Compare with Figure 3.2.

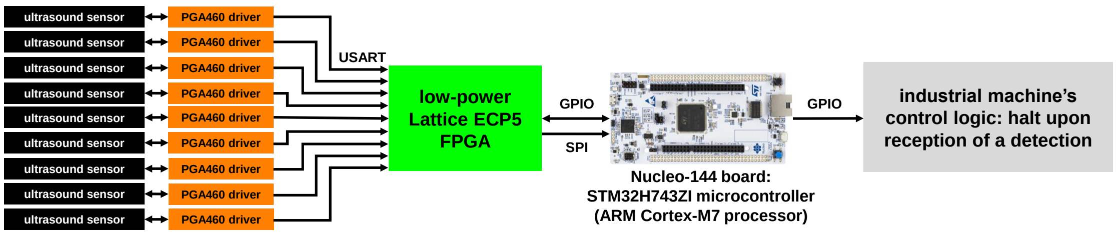
ultrasound sensor
PGA460 driver
USART
low-power Lattice ECP5 FPGA
GPIO
SPI
Nucleo-144 board: STM32H743ZI microcontroller (ARM Cortex-M7 processor)
GPIO
industrial machine's control logic: halt upon reception of a detection

Figure 3.4: Schematic of the system architecture. Sensor fusion on data acquired from 9 sensors is one of the key proposed improvements compared to the SoA [135].

just 1 is one of the key advances compared to [135]. Each transducer works both as an emitter and as a sensor for sound waves with a frequency between 30 kHz and 50 kHz; the sensing consists in emitting US pulses and receiving the echo reflected by obstacles. Each sensor is operated by a Texas Instruments PGA460, which integrates a low-noise amplifier, a programmable time-varying gain stage, a 12-bit ADC, and a DSP.[4] The configured ADC resolution was set to 8 bits, producing `uint8` data, which is a convenient format for a DNN quantized to 8 bits; the sampling frequency was set to 100kHz, and the sampling duration was set to 20.48ms-windows.

The low-power Lattice ECP5 LFE5U-85F FPGA[5] collects the data from all 9 sensors. It communicates with the sensors via USART, and configures the resolution, sampling rate, and sampling duration at start-up. Then, the FPGA transmits the package of 2048 samples $\times$ 9 channels 8-bit to the MCU via SPI. The motivation for using an FPGA for data aggregation is that the STM32H743ZI MCU does not have enough external interfaces; the FPGA allows to receive data from all 9 US sensors and convey them to the MCU through a single interface (i.e., the SPI).

The task of the MCU is to receive the data from the FPGA, run the DNN, and command the machinery to stop upon detection. The MCU is an STM32H743ZI[6] (2.2), mounted on a STM32 Nucleo-144 board.[7] This MCU mounts an ARM Cortex-M7 processor [30] with double-precision FPU operating at 480 MHz, 2 MB of Flash memory, 1 MB of SRAM (with 192 kB of tightly coupled scratchpad memory for real-time tasks), 4 DMA controllers, and peripherals such as UART/USART, SPI, Ethernet, and GPIO lines. Upon reception of the 2048 samples $\times$ 9 channels data, the MCU executes the DNN inference. If the outcome is positive, the MCU raises a GPIO connected to the controller of the industrial machine, which halts the machine.

All the listed hardware elements are commercial components. The motivation for this choice is that the purpose of this work is not to profile specific hardware elements

[4] https://www.ti.com/product/PGA460
[5] https://www.latticesemi.com/Products/FPGAandCPLD/ECP5
[6] https://www.st.com/en/microcontrollers-microprocessors/stm32h743zi.html
[7] https://www.st.com/en/evaluation-tools/nucleo-h743zi.html

but to test whether the task is viable with commonly available hardware. In particular, there is no need for high-precision ultrasound sensors since accurate acoustic waveforms are irrelevant for a binary detection task in the presence of acoustic noise. In general, different component choices are not expected to alter the prototype's performance in terms of latency and accuracy. Profiling or designing dedicated components is out of the scope of this work. Different component choices to adapt the system to specific use cases do not limit the conclusions of the methodology proposed in this work.

In a more optimized iteration of the system, the FPGA+MCU assembly can be avoided by either (*i*) deploying the TCN model onto the FPGA, removing the MCU, or (*ii*) replacing the FPGA with one or more commercial off-the-shelf ICs (or by an FPGA chosen to be as small and inexpensive as possible) performing the data aggregation, keeping the net on an MCU. The latter option has the advantage of programmability for specific use cases with environmental conditions so diverse and challenging to require to adapt more than the net's parameters, e.g., the net's structure or additional processing stages. However, this kind of optimization is out of the scope of this work since the FPGA+MCU assembly has enough performance to make the realized prototype an effective proof-of-concept at this applied research stage (as exposed in the results in Subsection 3.1.4).

3.1.3.3 Data acquisition

The dataset acquisition followed three criteria: (*i*) framing the ML application as a detection task, i.e., a binary classification task *presence-vs-absence* of an obstacle; (*ii*) create environmental conditions analogous to the ones of the industrial facilities where the target woodworking machine typically operates; (*iii*) collect enough data to allow for a good DNN's recognition accuracy even on data pertaining to diverse conditions. Time windows of US signals were collected with and without obstacles creating a US response echo; the different used obstacles were people, dummies, and wood panels, also in a joint fashion. In addition to the two classes presence-vs-absence of an obstacle, two varying conditions produced more diverse data representative of real variable working situations:

- obstacle-sensor distance, varied from 0.5 m to 2.0 m;
- application of a compressed-air jet, recreating the environmental noise of the machinery's room, varying the pressure level from 0.0 bar (i.e., no noise) to 3.0 bar and the jet-sensor distance from 0.5 m to 1.5 m, both in presence and in absence of an obstacle.

First, 5 collections of data were acquired without noise, then 3 collections with noise. In noisy acquisitions, the compressed-air jet was always on, and the pressure value was constant while running each acquisition. Subsection 3.1.4.1 reports the detailed structure of the signals and of the whole dataset.

3.1.3.4 Incremental learning protocol

Incremental learning on the dataset involved experiments with incremental splits of the noisy data, i.e., collections 6-to-8. In particular, collections 6, 7, and 8 were merged and randomly split into three blocks of equal size with stratification (i.e., the same proportion of collections in each block). These blocks are denoted as the noisy data's *first third*, *second third*, and *last third*. The incremental experiments use the following splits:

- **Experiment 0:** training on collections $1 \cup 3 \cup 5$ and validation on collections $2 \cup 4$; this experiment involves no noisy data and is a control on the acquisition system and the quality of the data;
- **Experiment 1:** training on noiseless data, and validation on the last $\frac{1}{3}$ of noisy data; this experiment measures how well a model can generalize to noisy data after only seeing noiseless data in training;
- **Experiment 2:** training on noiseless data plus the first third of noisy data, and validation on the last third of noisy data;
- **Experiment 3:** training on noiseless data plus the first and second thirds of noisy data, and validation on the last third of noisy data.

Experiments 1, 2, and 3 show the model progressively larger amounts of noisy data at training time; this allows assessing how much the proposed setup can benefit from incremental learning on newly-acquired data to improve detection. The validation set is the same across Experiments 1 to 3 for a fair comparison of the results. This diverse dataset and its incremental protocol are a key advance compared to [135], where the incremental learning scenario is simulated by mere aggressive augmentation up to $1000\times$ of a single collection of 227 single-channel signal windows (i.e., $22\times$ fewer examples than the 5085 acquired in this work).

It is important to remark that incremental training is not meant to be run in real-time: real-time is only required for inference, which is part of the online pipeline of acquisition-transmission-processing. When the operators desire new data to improve the detection under specific challenging conditions, the system can collect new data and store them to a server (e.g., via the MCU's Ethernet), which retrains the net by including the new data and sends the updated model parameters back to the MCU. This process is not meant to be real-time because the new acquisition and the retraining typically need human supervision and iterations. Typically, the bottleneck is not transmission or latency but resides in (*i*) data acquisition, which requires materially preserving or reproducing the conditions of interest, and (*ii*) the search for the training settings able to fit both the old and the new data. This process occurs at the time scale of human manual experimentation, not at the time scale of the online acquisition-transmission-execution pipeline.

3.1.3.5 TCN structure, training, and deployment

Temporal Convolutional Networks (TCNs) are a category of Convolutional Neural Networks (CNNs) specialized for time series. TCNs are based on 1D convolutions along the time dimension, and they outperform Recurrent Neural Networks on image segmentation, object detection, and biosignal processing [15], [21], [24], [27]. TCNs have also proven amenable to hardware-friendly parallelization strategies that reduce inference latency and energy consumption [16]. More details on TCNs are provided in Section 2.1.

The proposed TCN has 6 convolutional layers followed by 3 linear layers, and Table 3.2 reports the net's complete structure. The input $\mathbf{x}$ is a 2048 samples $\times$ 9 channels `uint8` US signal window produced as per 3.1.3.2 and 3.1.3.3. The 6 linear layers have 4, 4, 2, 2, 1, and 1 output channels, all with kernel size $k = 3$, full padding (i.e., zero-padding with length $p = 1$), and stride $s = 2$. The 3 linear layers have size 32-to-8, 8-to-8, and 8-to-1; the final scalar represents the input's score $\hat{y}_{\text{soft}} = \text{TCN}(\mathbf{x}) \in [0, 1]$, which is the soft (i.e., not yet binarized) assignment for the binary classification. All layers have BN and ReLU activation except the last linear layer, which flows into a sigmoid. After training, BN folding is applied to merge each BN with its previous layer, slightly reducing the number of parameters and operations.

This TCN has just 560 parameters and requires just $151 \cdot 10^3$ MAC operations; The activation memory footprint is the maximum consecutive activation maps, i.e., input and output of a single layer; this is reached in the first convolutional layer with $22.5 \cdot 10^3$ activations (9×2048 input plus 4×1024 output. With 8-bit quantization (explained in the next paragraphs), the parameters and activations memory footprints amount to 560 B and 22.0 KiB, respectively. This size makes the net very hardware-friendly for resource-constrained embedded platforms for computation and memory requirements. Moreover,

Table 3.2: Detailed structure of the proposed TCN, including the breakdown of all layers' memory footprint and computational load. All layers are sequential in a feed-forward fashion so that each layer's output format is the input format of the next one. As to sizes, the numbers of tensor elements directly correspond to the memory occupancy in bytes, thanks to 8-bit quantization. The field "# MAC" refers to the number of Multiply-and-Accumulate (MAC) operations.

	LAYER	**Size**			**# MAC**	
		input	**parameters (weights + biases)**	**output**		
1	**convolutional**	9×2048	112	4×1024	114688	(76.1 %)
2		4×1024	52	4×512	26624	(17.7 %)
3	kernel $k = 3$	4×512	26	2×256	6656	(4.4 %)
4	padding $p = 1$	2×256	14	2×128	1792	(1.2 %)
5	stride $s = 2$	1×128	7	1×64	448	(0.30%)
6		1×64	4	1×32	128	(0.08%)
	(flattening)	1×32	0	32	0	
7	**fully connected**	32	264	8	264	(0.18%)
8		8	72	8	72	(0.05%)
9		8	9	1	9	(0.01%)
	COMBINED	Total parameters: 560 Max consecutive maps: $22.5 \cdot 10^3$ (layer 1)			Total MAC: $151 \cdot 10^3$	

it directly processes the raw signals without any handcrafted feature extraction or preprocessing, thanks to automatized feature learning at training time: this avoids time-consuming feature engineering and computation latency before inference.

Training consisted in 2 epochs in `float32`, followed by Post-Training Quantization (PTQ) to 8 bit and 16 epochs of Quantization-Aware Training (QAT) (2.3.1). Quantization to 8-bit reduces the parameters memory requirement to 560 B and the activations memory requirement to 22.0 KiB, which are both $\frac{1}{4}$ of their `float32` counterparts. Both stages of training used balanced binary cross-entropy loss, Adam optimizer, initial learning rate 10^{-4}, and minibatch size 64. Both PTQ and QAT used the technique of PArameterized Clipping acTivation (PACT) [67]. Both trainings exploited the augmentation of the training set by a factor 64×, which consisted in producing 64 altered versions from each original US window by applying two transformations:

- a scaling by a factor from a uniform random distribution on $[0.95, 1.05)$, followed by casting back to `uint8`;
- a temporal shift by a random amount from a uniform distribution on $\{-25, \ldots, +25\}$ samples.

This augmentation scheme is similar to [135]; still, the advances of this work allow to achieve accurate detection with 15× milder augmentation (i.e., 64× instead of 1000×), thanks to the inherent richness of the novel dataset (3.1.3.3). In this setup, the sources of randomness are augmentation, net initialization, and stochastic minibatching. So, each of the Experiments 0, 1, 2, and 3 involved 64 repetitions to get statistics about the detection metrics.

The TCN was implemented using Python 3.8, PyTorch 1.9.0 [65], [66], and the open-source quantization library QuantLib (2.3.2). The TCN quantized to 8 bit was exported in ONNX format (2.3.2) and deployed onto the STM32H743ZI MCU using the environment STM32CubeIDE 1.12.0 for code generation and exploiting X-CUBE-AI 8.0.0 (2.3.2), the software extension for configuring DNN inference execution on STM32 MCUs using ARM CMSIS kernels [58]. The stages in STM32CubeIDE or X-CUBE-AI did not include any further quantization or compression.

3.1.3.6 Evaluation metrics

This work targets both classification metrics, which measure the correctness of the TCN's detection, and deployment metrics, which quantify the computation and resource budget required by the TCN on the STM32H743ZI MCU.

The addressed classification metrics are the ones typical of detection (i.e., binary classification) on unbalanced data:

- *sensitivity* (synonym of *True Positive Rate* (TPR) or *recall*): the fraction of actual positives correctly detected:

$$\text{sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}; \tag{3.1}$$

- *specificity* (synonym of *True Negative Rate* (TNR)): the fraction of actual negatives correctly classified:

$$\text{specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}; \tag{3.2}$$

- *balanced accuracy* (synonym of *macro-average accuracy*): the average of sensitivity and specificity.
- Area Under the Receiver Operating Curve (AUROC).

All these metrics are independent of the class imbalance in the data, as opposed to naïve unbalanced accuracy. The pair sensitivity-specificity provides a more complete characterization than the pair precision-recall often used for binary classification since the latter pair does not consider the number of True Negatives; in contrast, the former pair considers all four possible outcomes. As to AUROC, it is independent of the threshold Θ used to determine the estimated hard labels $\hat{y}_{\text{hard}} \in \{0, 1\}$ from the $\text{TCN}(\cdot)$ model's output soft labels $\hat{y}_{\text{soft}} \in [0; 1]$ for each input $\mathbf{x}$:

$$\hat{y}_{\text{hard}} = \begin{cases} 1 & \text{if } \hat{y}_{\text{soft}} = \text{TCN}(\mathbf{x}) > \Theta \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

Thus, the AUROC is methodologically interesting because it allows assessing the detection correctness independently from the specific sensitivity-specificity tradeoffs fixed in different application use cases. Since sensitivity and specificity depend on the choice of the discrimination threshold, the reported results refer to the threshold values tuned to maximize the balanced accuracy to report an example of tradeoff.

The addressed deployment metrics profile the workload of the real-time on-edge computation: memory footprint of the model; latency per inference; power consumption of inference, measured in working conditions $f_{\text{clock}} = 480\,\text{MHz}$ and $V_{\text{dd}} = 3.3\,\text{V}$, via a USB power meter averaging over $30\,\text{s}$ while executing inferences in loop; and energy per inference, determined as power$\times$latency.

3.1.4 Experimental Results

3.1.4.1 Dataset

Figure 3.5 and Figure 3.6 show the typical behaviour of the acquired signals. All windows begin with the final segment of the US burst, which saturates the ADC's `uint8` dynamic range and carries no information about the class. However, this segment is a valuable control for diagnostics since it always presents the same timing across different recordings. The initial saturation in each sensor's data only comes from the final segment of the US burst of that sensor. This check consisted in the following experiment. The procedure to check whether the burst from sensor i affected sensor $j \neq i$ involved running obstacle-less, noise-less runs starting sensor i's acquisition 10 ms after sensor j's acquisition. This means that sensor i's burst emission happens between time 0 ms and time 10 ms of sensor j's acquisition (which is 20.48 ms in total, as per 3.1.3.2). So, if cross-sensor interference is present, it is visible in the first half window of sensor j's data. Looping i and $j \neq i$ over all 9 sensors showed no cross-sensor interference for any (i, j) pair. This means that the adopted sensor placement causes no interference across sensors in the burst emission stage.

Later in the window, after the initial saturation due to the final segment of the emitted US burst, the echo carries the information of interest. As shown in Figure 3.6, the noise resulting from a compressed air jet strongly affects the pattern of the echo signal envelopes. This makes it hard to devise handcrafted features that intuitively discriminate obstacle echoes from intensity due to noise, making classification hard, especially concerning specificity and the occurrence of false positives. This confirms the motivation for the recourse to DNNs (particularly TCNs) capable of automatic feature-learning at training time to obtain a data-driven feature extraction based solely on optimizing detection accuracy.

The realized dataset has the structure reported in Table 3.3. The whole dataset consists of 8 *collections*, and each collection is composed of 10 to 30 *runs*, for a total of 170 runs. The choice of the terms *collections* and *runs* is to avoid ambiguous naming such as *acquisitions*, *samples*, or *sessions*. Each collection corresponds to a value of the distance of the compressed-air jet, when applied; globally, the dataset contains 5 collections without noise and 3 collections with pressure noise. Within each collection, the different runs correspond to a choice of the obstacle-sensor distance and the jet's pressure and orientation (if applied). Between runs, the whole system was turned off and on. So, runs are homogeneous subsets of the dataset since they contain 2048-sample windows acquired in identical conditions of all settings, namely obstacle-sensor distance, compressed air jet pressure, and compressed air jet distance. Each collection

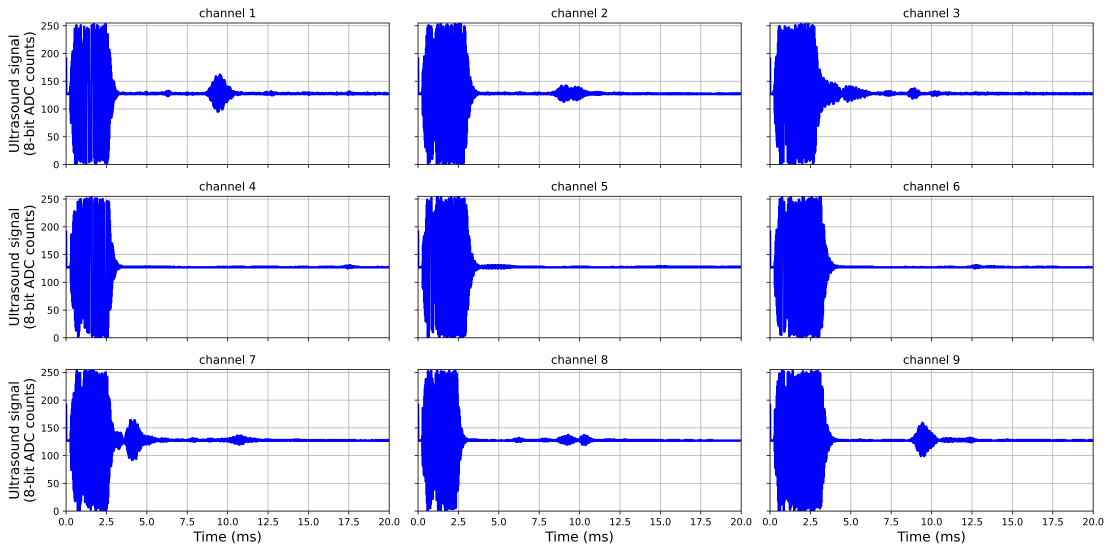

Figure 3.5: Example of a US window with obstacles and without noise (collection 1, run 10, window 1; all 9 channels, all samples except the last 48). It is possible to see the initial US burst, the subsequent silence, and the echoes received by the sensors facing obstacles.

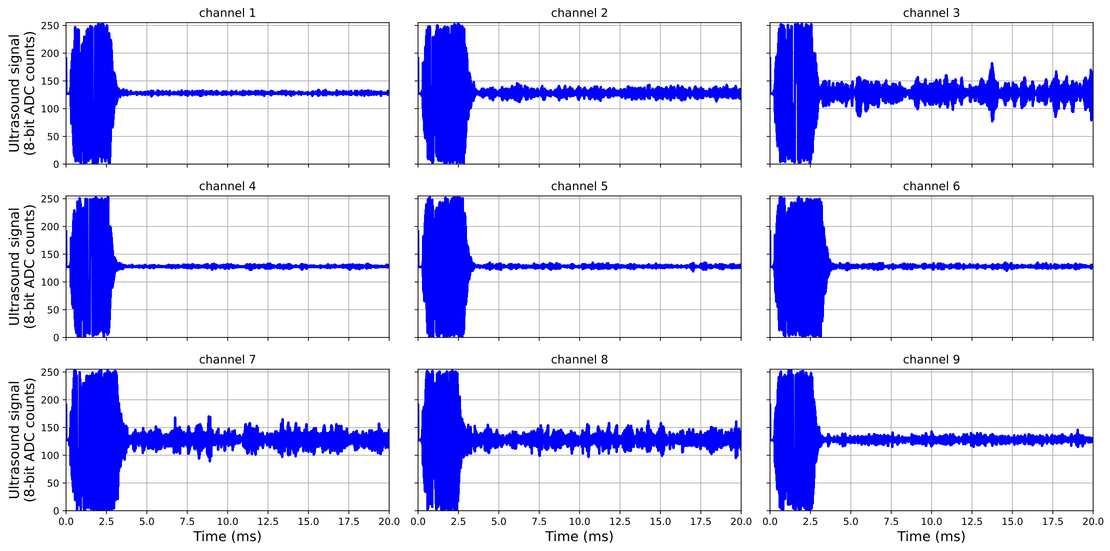

Figure 3.6: Example of a US window without obstacles and with noise from the compressed air jet (collection 8, run 50, window 1; all 9 channels, all samples except the last 48). The sensors most affected by noise sense an amplitude comparable to obstacles' echoes in the absence of noise (Figure 3.5).

consists of runs acquired with the same compressed air jet distance (if present), hence containing 2048-sample windows that are diverse due to varying obstacle-sensor distance and compressed air jet pressure (if present).

Table 3.3: Dataset of ultrasound windows realized for setup validation and incremental learning. The dataset consists of *collections*; in turn, every collection contains *runs*. Each run contains data acquired with the same obstacle-sensors distance, and the air jet pressure (if present) and air jet distance. Each collection contains runs corresponding to different obstacle-sensors distances and air jet pressures, but the same air jet distance.

COLLEC-TION	**Environment conditions**	**Number of runs**	**Windows per run**	**Total windows in collection**	**Negative class**		**Positive class**		**Class-imbalance of windows (neg – pos)**
					runs	**windows**	**runs**	**windows**	
1	no noise	10	30 (15 in run 9)	285	2	45	8	240	15.8% – 84.2%
2		16	30	480	3	90	13	390	18.8% – 81.3%
3		30	30	900	10	300	20	600	33.3% – 66.7%
4		20	30	600	20	600	0	0	100.0% – 0.0%
5		24	30	720	24	720	0	0	100.0% – 0.0%
6	compressed-air jet noise at 1.0 – 1.5 m	20	30	600	0	0	20	600	0.0% – 100.0%
7		25	30	750	25	750	0	0	100.0% – 0.0%
8	compressed-air jet noise at 0.5 m	25	30	750	25	750	0	0	100.0% – 0.0%
All		170		5085	109	3255	66	1830	64.0% – 36.0%

3.1.4.2 Accuracy

Figure 3.7 and Table 3.4 report the detection results of the four experiments conducted as per 3.1.3.4. Statistics are computed over the 64 repetitions of each experiment, performed to account for the fluctuations introduced by the sources of randomness in the process, namely data augmentation, initialization of the net's parameters, and mini-batching for stochastic gradient descent, as explained in 3.1.3.5. Figure 3.7 shows that the experimental distributions obtained for the detection metrics are highly skewed, as can be seen from the asymmetric IQR ranges, whiskers, and outliers; therefore, median $\pm$ Mean Absolute Deviation (MAD) is a convenient choice for summarizing each experiment in a way that is more robust and less sensitive to skewness compared to average $\pm$ standard deviation. The MAD is defined as

$$\text{MAD} \triangleq \text{median}_i\left(|a_i - \tilde{a}|\right) \tag{3.4}$$

where a_i's are the accuracy values of a single repetition, and $\tilde{a}$ is the experiment's median. It is worth remarking that, due to the non-linearity of the median, the median balanced accuracy is not the average of median sensitivity and median specificity, in general. The next paragraphs expose the results of Experiments 0, 1, 2, and 3 (structured as per 3.1.3.4), discussing each experiment individually.

Experiment 0 Experiment 0 is based on noiseless data for both training and validation (details in 3.1.3.4). Therefore, this experiment is a check for the setup and the produced data. The outcome of this experiment is positive since all detection metrics (namely sensitivity, specificity, balanced accuracy, and AUROC – explained in 3.1.3.6) have a median $> 97\%$. For instance, these results show a key successful sanity-check in that the working surface (Figure 3.1, Figure 3.2, and Figure 3.3) is correctly discriminated from the added obstacles, despite being itself a physical object in the sensors' field of view.

Experiment 1 Experiment 1 consists in training on noiseless data and validation on noisy data (details in 3.1.3.4). This experiment yields a balanced accuracy and an AUROC collapsed to values compatible with the chance level, which is $\frac{1}{2}$ for both these detection metrics. This collapse shows that recognition of noisy data is impossible if the model has never seen data affected by the compressed air jet pressure at training time; this confirms the motivation of the chosen protocol for data collection and incremental learning.

Table 3.4: Detection metrics results for Experiments 0, 1, 2, and 3 (protocol detailed in 3.1.3.4). Distributions are summarized as median ± Mean Absolute Deviation (MAD). This chart complements Figure 3.7 by reporting quantitatively the high accuracy of Experiment 0, the collapse in Experiment 1, and the recovery in Experiments 2 and 3.

	Detection metric (median ± MAD)			
	sensitivity	**specificity**	**balanced accuracy**	**AUROC**
Experiment 0	0.967 ± 0.018	0.991 ± 0.006	0.980 ± 0.007	0.993 ± 0.002
Experiment 1	0 ± 0	1 ± 0	0.5 ± 0	0.493 ± 0.016
Experiment 2	0.850 ± 0.055	0.964 ± 0.028	0.889 ± 0.028	0.936 ± 0.031
Experiment 3	0.905 ± 0.030	0.952 ± 0.028	0.917 ± 0.024	0.972 ± 0.013

Experiment 2 Experiment 2 adds $\frac{1}{3}$ of the noisy data to the training set (details in 3.1.3.4). The results of this first step of incremental learning on noisy data show that the detection in the presence of noise strongly surpasses the chance level, yielding a sensitivity of $85.0 \pm 5.5)\%$ and all other metrics $> 88\%$. This experiment crucially proves that the data contain a pattern also in the presence of noise and that this pattern is strong enough to allow for an accurate data-driven detection.

Experiment 3 Experiment 3 adds a further $\frac{1}{3}$ of the noisy data to the training set (details in 3.1.3.4). In this experiment, all the detection metrics except specificity further increase compared to Experiment 2. Specificity stays constant since it only decreases by 1.2%, and the new value is consistent with Experiment 2 within the variability MAD = 2.8%. This experiment proves that the proposed system and DL setup are able to leverage increasing amounts of data to improve its accuracy on the challenging real working conditions of the industrial facility's environment.

Discussing detection metrics with an end-to-end view requires explaining what happens if the proposed system fails to detect an obstacle. In this case, a collision can happen between the obstacle and a machine's soft bumper; this kind of collision is not dangerous since bumpers are part of the active safety system that stops the machine in case of contact (as explained in 3.1.2.1). In general, it is possible to create even more redundancy by combining the proposed systems with any of the existing SoA active safeguards illustrated in 3.1.2.1, such as pressure-sensitive floor mats, photocell barriers, or laser scanners.

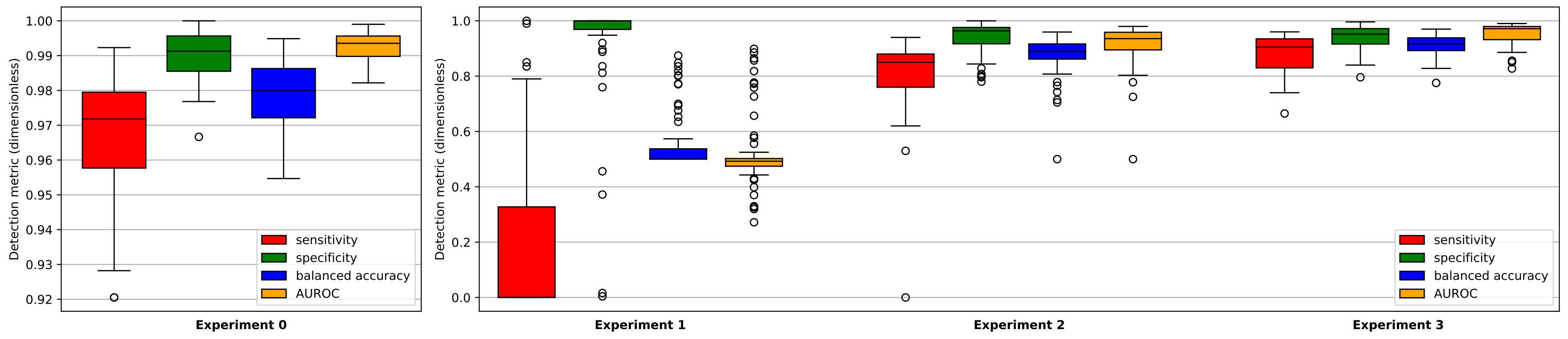

Figure 3.7: Experimental distributions of the detection metrics obtained for Experiment 0 (validation of setup and data) and Experiments 1-to-3 (incremental training on noisy data); for the details of the experimental protocol, see 3.1.3.4. Notice the different y-scales in the two plots. The lower (resp., upper) whisker is set at the lowest datum above $Q_1 - 1.5$ IQR (resp., $Q_3 + 1.5$ IQR), with Q_1 and Q_3 the first and third quartiles respectively, and IQR $\triangleq Q_3 - Q_1$ the interquartile range. The general trend shows high accuracy in Experiment 0, the collapse in Experiment 2, and the incremental recovery in Experiments 2 and 3. Moreover, the asymmetric IQR ranges, whiskers, and outliers highlight high skewness; this motivates the recourse to median ± Mean Absolute Deviation (MAD) for more robust summaries compared to average ± standard deviation.

3.1.4.3 Performance and memory footprint

Table 3.5 reports the results of the TCN profiling, compared with [135]. For a fair comparison, since [135] dealt with just 1 input channel, that net is also extended to support 9 input channels as the new data. Memory footprints refer to TCN quantized to 8 bit. Memory footprints of activations are determined as the maximum sum of two consecutive feature maps since batch normalizations and ReLUs can be computed in place; for all models, the maximum-size pair is the input-output of the first convolutional layer, which occupies $(C_{\text{in}}T_{\text{in}} + C_{\text{hid 1}}T_{\text{hid 1}})$ bytes, where $T_{\text{in}} = T_{\text{hid 1}} = 2048$ samples, C_{in} is 1 channel for [135] and 9 channels for extended-[135] and the proposed new net; and $C_{\text{hid 1}}$ is 2 channels for [135] and extended-[135], and 4 channels for the proposed new net.

The energy consumption per inference was determined based on the power draw measured experimentally, which is (1.63 ± 0.01)W, which is in the same range as the previous work [135]. Overall, the results show that the proposed new TCN improves all the deployment metrics, except the RAM used for activations, which is the same as the reference model, i.e., 22.0 KiB (2.1% of the total 1 MiB available on the STM32H743ZI MCU). The advantage of the new compact model lies in the latency and energy consumption per inference reduced by $> 2.27\times$ compared to [135].

It is essential to note that a latency of $\Delta t_{\text{infer}} = 5.0$ ms/inference does not imply a rate of $1/\Delta t_{\text{infer}} = 200$ inferences/s. In general, the entrance of operators or objects into the space spanned by the moving parts of the machine (corresponding to the field of view of the sensors) can be detected using an inference rate much lower than 200 inferences/s. For specific applications, the inference rate choice is based on the use case's requirements. Moreover, a higher inference rate gives some degrees of freedom for postprocessing operations such as majority voting or averaging of the scores to make accuracy more robust. In situations that do not require a high inference rate, using an MCU with lower performance is possible, continuing to satisfy the real-time requirements.

It is worth discussing the latency results in more detail. The speed v_{cabinet} of the machine's moving parts is of the order of 1 m/s (as explained in 3.1.2.1), and the speed v_{obs} of potential obstacles, i.e., people and objects in the surroundings, is typically lower. Assuming the worst case, i.e., the machine's moving cabinet and an obstacle moving towards each other from a distance d, the maximum allowed stopping time T_{max} is

$$T_{\text{max}} = \frac{d}{v_{\text{cabinet}} + v_{\text{obs}}}. \tag{3.5}$$

A conservative estimate of T_{max}, which corresponds to a conservative upper bound on latency, can be obtained assuming $v_{\text{cabinet}} = 1$ m/s, $v_{\text{obs}} = 1$ m/s, and $d = 0.5$ m, which

Table 3.5: Results of the profiling of the proposed TCN's deployment and execution.

MODEL		**Memory**		**Operations (kMAC)**	**Latency (ms)**	**Energy per inference (mJ)**
		weights (B)	**activations (KiB)**			
Conti et al. [135]	original: 1 input channel	1044	6.0	227	11.4	18.6
	expanded to 9 input channels	1792	22.0	630	> 11.4	> 18.6
This work		560 (0.32×)	22.0 (same)	151 (0.24×)	5.0 ($< 0.44\times$)	8.2 ± 0.2 ($< 0.44\times$)

is the shortest distance used in the dataset (3.1.3.3). These values yield

$$T_{\max} = \frac{0.5\,\text{m}}{1.0\,\text{m/s} + 1.0\,\text{m/s}} = 0.25\,\text{s}. \tag{3.6}$$

This underestimate of the maximum allowed latency is 12× the acquisition time of the signal, i.e., 20.48 ms (3.1.3.2), and 50× the computation latency of the TCN inference, i.e., 5.0 ms. The US sensors can detect obstacles at a maximum distance of 2 m to 2.5 m, so more time is generally available. Even in the worst case, the proposed system's latency for data acquisition and processing is one order of magnitude shorter than the available time: the proposed solution has a latency sufficiently short for the task, with a significant margin for future scenarios with faster-moving cabinets and obstacles.

3.2 Low-Latency Detection of Epileptic Seizures from iEEG with Temporal Convolutional Networks on a Low-Power Parallel MCU

3.2.1 Overview

Although the primary treatment for epilepsy is pharmacological, approximately one-third of patients are affected by drug-resistant forms of epilepsy [144]. These cases can either require surgical treatment [145] or benefit from closed-loop brain stimulation [146]. The latter can eliminate or mitigate the seizure symptoms and relies on coupling a neuromodulator with a real-time detection system that recognizes the onset of seizures based on the analysis of the brain signals. Closed-loop neuromodulators are implantable devices that read intracranial Electro-Encephalographic (iEEG) signals and stimulate the brain tissue, and implantability imposes very strict computational resources and energy budget.

Currently, the iEEG signal allows the best spatial resolution and provides the highest signal-to-noise ratio compared to other neural recording techniques [147]. With this biosignal, many attempts have been made to develop frameworks to detect seizures. Recently, several works have proposed methods based on Machine Learning [148]–[150] and Deep Learning [151], [152] to successfully detect the *ictal* (i.e., during a seizure) and the *inter-ictal* (i.e., between seizures) states from the iEEG signal. High *sensitivity* and *specificity*, and short *delay* (i.e., the time between the onset and the recognition of a seizure) are fundamental parameters for evaluating the quality of an epilepsy detection system. Above all, *specificity* is critical because studies have shown that false positives can generate high levels of anxiety and stress in patients [146]. Hence, they must be minimized.

For the automated learning approach, an invaluable source of brain activity data is the iEEG recorded in Epilepsy Monitoring Units (EMU), where it is possible to perform pre-surgical long-term observations. Typically, EMU patients are monitored for only 1 to 3 weeks to minimize the discomfort and the risk of adverse effects (e.g., infection and inflammation deriving from the iEEG electrodes implanted through the skull) [153]. The collected data are essential for preliminary monitoring to plan personalized surgical treatment, but they are also used as a base for training algorithms for real-time seizure recognition [154]. Given the highly patient-specific nature of seizure dynamics, seizure detection frameworks require tuning to each patient [155]. This patient-dependent approach poses significant challenges because of the highly imbalanced nature of the data, where inter-ictal states are much longer than ictal states (class-imbalance problem).

A major challenge in real-time seizure detection is to design computationally efficient frameworks that can provide reliable recognition while at the same time meeting the strict computation, memory, and power constraints of embedded platforms working in real time. In this section, I address the problem of iEEG-based detection of epileptic seizures in real-time, targeting the Short-Term SWEC-ETHZ iEEG Database [154]. I present a solution based on a TCN designed for low-power edge monitoring platforms. I present the following contributions:

- I present a novel TCN network with 1D dilated convolutional layers, enabling a more efficient pattern extraction from input time windows; this yields a compact model requiring just 2.52 kB of memory footprint and 164 kMAC of computation, working entirely at `int8` bitwidth; the obtained detection delay is up to 10 s shorter than the SoA setup based on Hyper-Dimensional Computing (HDC); at the same time, the model satisfies the same sensitivity and specificity constraints as the SoA; furthermore, the setup is time-consistent: training seizures always precede testing ones temporally, a constraint which would be present in the clinical practice, but that is unfortunately not taken into account in the SoA work [154].
- I deploy the model on the low-power edge microcontroller GAP8 [33], [34], attaining a computation latency of just 5.68 ms and an energy cost of just 124.5 µJ when executed on 1 core, and latency 1.46 ms and an energy cost 51.2 µJ when distributed on 8 cores. These values are better than the HDC SoA [154] and are a perfect fit for long-term monitoring by an embedded SoC working in real time.

3.2.2 Materials & Methods

3.2.2.1 Short-Term SWEC-ETHZ iEEG Database

Intracranial Electroencephalography (iEEG) is an invasive technique to acquire brain signals via electrodes implanted surgically directly onto the surface (strip-, grid electrodes) or even into the brain (depth electrodes) [156]. Compared to extracranial EEG, the iEEG provides better spatial and temporal resolution (mm-scale and ms-scale, respectively [157]), higher bandwidth, less noise, and fewer artifacts, though with the drawback of requiring surgery with a higher risk of infection [158]. The dataset addressed in this work is the Short-Term SWEC-ETHZ iEEG Database [154], a publicly available[8] iEEG dataset containing epileptic seizure recordings from 16 patients of the epilepsy surgery program of the Inselspital Bern, for a total of 100 seizures. The number of seizures varies from 2 to 14 across patients.

The iEEG signals were acquired by either implanted strip, grid, and depth electrodes or by a mixed configuration of these electrode types. Electrode numbers (varying from 36 to 100 across subjects) and implantation schemes were established based on clinical needs. An extracranial electrode localized between the `Fz` and `Cz` positions (*10-20 system*) was used as reference. The sampling rate was either 512 Hz or 1024 Hz, depending on whether each patient had more or less than 64 electrodes implanted. Prior to further analyses, the signals recorded with less than 64 electrodes were downsampled to 512 Hz. All signals were re-referenced against the median of all electrodes free of permanent artifacts (e.g., 50 Hz PLI), as judged by visual inspection [159]. The signals were digitalised to 16 bit and band-passed with a 4th-order Butterworth filter with band 0.5 Hz–150 Hz. For seizure onset marking, which constitutes the dataset's ground truth, the iEEG traces were visually inspected by an experienced board-certified epileptologist [154]. Electrodes permanently corrupted by artifacts were excluded by the same procedure. The dataset's ictal segments range from 10 s to 1002 s. In addition, each recording includes 180 s of the inter-ictal state preceding the seizure and 180 s of the post-ictal state.

3.2.2.2 TCN Framework

This contribution addresses epileptic seizure detection by treating the iEEG signal as a time series, applying a TCN based on the SoA model EEGNet [160], a CNN specialized for EEG. EEGNet has proven powerful on several tasks ranging from the classification of steady-state visual evoked potentials [161] to motion imagery recognition [162]. On

[8] `http://ieeg-swez.ethz.ch`

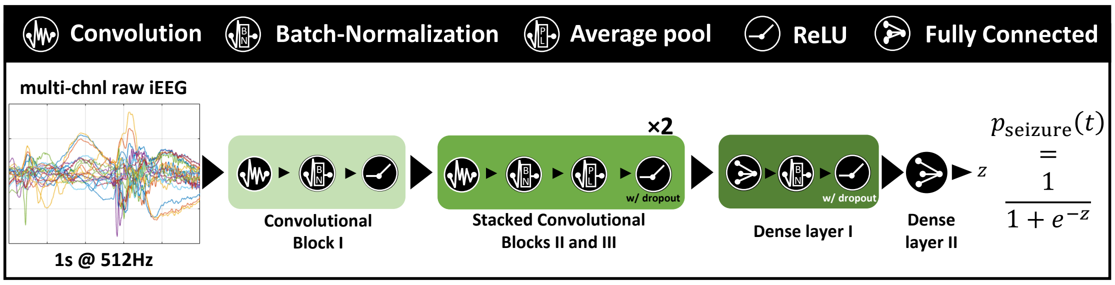

Figure 3.8: The TCN used in this work, inspired by EEGNet [160], [161].

top of these results, the EEGNet topology is exploited as a base to design a TCN that matches the memory and computation constraints of low-power edge microcontrollers. In particular, the block structure is preserved while applying a reduction of parameter number.

The EEGNet-inspired TCN is shown in Figure 3.8. It has 3 Convolutional Blocks each composed of 4 filters with BN [117]:

- Convolutional Block I has a unit kernel, i.e., $k = 1$, since it is in charge of the *spatial filtering* in the EEG sense: it mixes the input iEEG electrodes (spatially distributed) into new network channels, with a time-independent linear combination;
- Convolutional Blocks II and III extract the temporal information performing dilated causal convolutions with $k_{\text{II}} = k_{\text{III}} = 3$, and dilation $d_{\text{II}} = 2$ and $d_{\text{III}} = 4$.

Finally, two stacked 32-unit dense layers compute the probability of the input window belonging to the ictal state, which is returned by a sigmoid activation. Overall, this TCN has 2520 parameters and executes 164 kMAC.

3.2.2.3 Baseline and time-consistent setup

Inspiring works have addressed the Short-Term SWEC-ETHZ iEEG Database with automated learning [154], using models both from classical ML (RF, SVM, MLP, and HDC) and from DL (2D-CNN and LSTM). The current SoA algorithm on the targeted dataset is an HDC Ensemble applied on 3 signal features, namely Local Binary Pattern, Line Length, and Amplitude, which provides a seizure detection with a specificity of 97.3 % and a detection delay of 8.81 s while missing only 3.6 % of the dataset's seizures [154]. In particular, Table 3.6 details how the HDC Ensemble stands out as the SoA baseline by attaining better specificity and detection delay compared to deep models such as 2D-CNN and LSTM, at a comparable miss rate.

Table 3.6: Summary of the results of [154], indicating the HDC Ensemble as SoA baseline on the Short-term SWEC-ETHZ iEEG Database, against other deep models.

Model	**Missed seizures**	**Specificity**	**Detection delay**
HDC Ensemble	3.6%	**0.973**	**8.81s**
STFT + 2D-CNN	**2.3%**	0.836	17.9s
raw iEEG + LSTM	4.7%	0.948	14.7s

A limitation of all the works above on the Short-Term SWEC-ETHZ iEEG dataset is that training and test seizures are not temporally consistent: training seizures do not always precede testing ones. This is because the focus of all the cited approaches is determining the minimum number of training seizures required to have a good recognition on unseen seizures (*few-shot learning*), regardless of chronological order. In contrast, this work is interested in time consistency in this work. Hence, this contribution performed the training on the first half of each patient's seizures and the test on the second half. Note that in clinical practice, the training would indeed be performed on EEG traces and seizures that happened in the past, with the goal of detecting future ones. Though epileptic seizures of an individual patient are generally considered to be very similar, in a recent landmark study, Schroeder *et al.* have reported that they found significant variability in seizure evolutions, with more similar seizures occurring closer together in time [163]. Furthermore, others have observed that seizure patterns and severity may change under conditions such as pre-surgical evaluation when anti-epileptic drugs are often rapidly tapered to provoke seizures and thus shorten the time needed to obtain enough information for a decision about the feasibility of surgically removing the epileptogenic brain regions [164]. To perform a fair comparison, this work applies the time-consistent training setup to both the SoA HDC approach and the proposed EEGNet-inspired TCN.

3.2.2.4 Details on the ML setup

Timing Both the HDC Ensemble and the TCN are fed with 1 s-windows of the multichannel iEEG signal at 512 Hz. The HDC performs feature extractions as described in 3.2.2.3, whereas the TCN directly executes convolutions on the raw signal. For training, windows are taken with a slide of 0.5 s for HDC and 32 ms for the TCN; at inference time, the slide is 0.5 s for both algorithms, thus delivering 2 inferences/s.

TCN training The TCN (implemented in PyTorch 1.6 [65], [66]) was trained with binary cross-entropy loss, AdaM optimizer, initial learning rate 0.001, and minibatch size 64, for 15 epochs in `float32` plus 1 epoch in `int8` (quantization details in 3.2.2.4).

Postprocessing & delay-specificity curves Postprocessing was applied to both the HDC's and TCN's outputs using a n-sample checker, as per [154]: after each inference, a window of n model outputs (0.5 s apart, as detailed in 3.2.2.4) in the past is considered, and a positive label is returned only if all n are positive. By using different values of n (starting from $n = 1$, i.e., no postprocessing), the experiments explored different trade-offs between specificity and detection delay (which will be defined in 3.2.2.5). A smaller n means a shorter window, hence a milder attenuation of positives, thus prioritizing high sensitivity and short detection delay over specificity. Conversely, a larger n takes a longer window, leading to a stronger attenuation of positives, yielding higher specificity at the cost of lower sensitivity and longer detection delay. In this way, the experiments characterized the HDC and TCN in terms of specificity-delay curves.

TCN quantization & deployment After training the TCN in `float32` format for 15 epochs, the pipeline included 8-bit PTQ and 1 further epoch of QAT to recover accuracy (2.3.1). The quantization method applied was the PArameterized Clipping acTivation (PACT) [67], as implemented in the open-source library NeMO (2.3.2) [43], [64]), developed to minimize CNN network memory footprint and latency to enable implementation on resource-constrained ultra-low-power platforms. In particular, quantizing the 2520-parameter model from `float32` to `int8` cuts its memory footprint by 4×, from 10.08 kB to 2.52 kB. The `int8`-TCN was deployed on the PULP microcontroller GAP8 (2.2) [33], [34], to measure the inference latency and the energy cost per inference. This deployment was done by exploiting the open-source tool DORY (Deployment ORiented to memorY) (2.3.2) [41], [42], using an extension to the backend to enable the support of dilated convolutional layers.

3.2.2.5 Evaluation Metrics

The detection of epileptic seizures was evaluated using three metrics, following the standard of the previous works on the dataset [154]:

- *sensitivity*: the fraction of detected seizures for each patient; seizures are detected when the classifier returns at least 1 True Positive inference, i.e., at least 1 positive inference over the ictal segment ($t > 180.0$ s in all recordings); note that this sensitivity is defined per-patient as a count over seizures, not over single inferences within a single seizure;
- *specificity*, defined as the fraction of True Negative inferences over the inter-ictal segment ($t < 180.0$ s in all recordings);

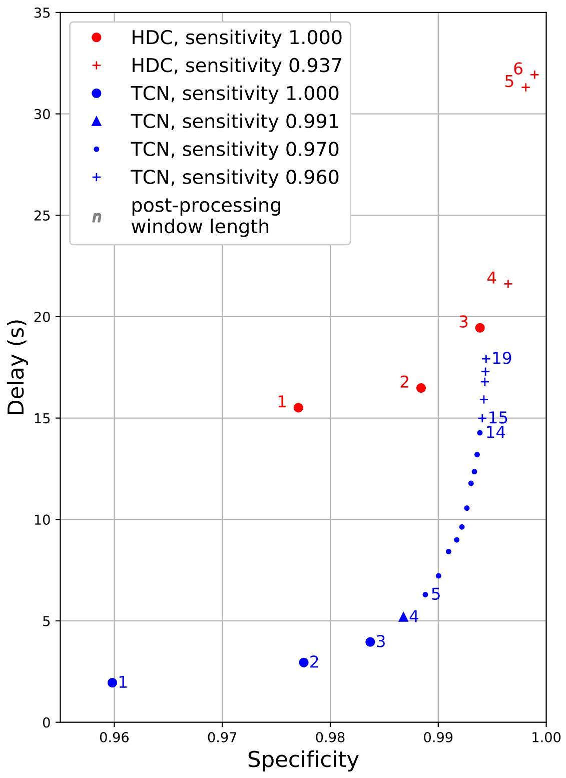

Figure 3.9: Delay-specificity curves of HDC and TCN, obtained by applying a different amount of postprocessing that inhibits positives.

- *detection delay*, measured as the time distance between the ground-truth seizure onset (at instant $t_0 = 180.0\,\text{s}$ in all recordings) and the first True Positive inference; following the definition in the baseline work [154], undetected seizures are discarded from the calculation of the average delay; doing so is fair as long as sensitivity is high, i.e., very few of the 100 seizures included in the dataset are missed.

Note that the detection delay is distinct from the computation latency to execute the model inference; the latter proves negligible compared to the former, as explained in the results in 3.2.3.2.

3.2.3 Experimental Results

3.2.3.1 Delay-specificity Pareto frontier

The recognition results of the TCN and the baseline HDC Ensemble are shown in Figure 3.9. The plot displays the specificity-delay curve obtained for each model by varying n for the n-sample checker, i.e., the length of the postprocessing window, as explained in 3.2.2.4. With no postprocessing (i.e., $n = 1$), all the positive outputs are retained, leading to the configuration that most favors a short detection delay over higher specificity. Increasing the number of samples used for postprocessing removes a higher fraction of positives, shifting the tradeoff toward higher specificity at the cost of an increased detection delay. Since the average detection delay is well-defined only at high sensitivity, i.e., when few seizures are missed (as discussed in 3.2.2.5), the curve points are considered valid only for sensitivity > 0.93, and the upper-right end of the curves is stopped at this threshold.

Remarkably, the TCN is able to provide a shorter detection delay when specificity is in the interval $[0.975, 0.995]$. Thus, the proposed TCN constitutes the Pareto frontier in this region. Furthermore, if the sensitivity requirement is raised above 0.95, the Pareto frontier is entirely represented by the TCN only. This is because the high-specificity points of the HDC curve have sensitivity 0.937, whereas the high-specificity points of the TCN curve always have sensitivity $\geq$ 0.96. The TCN's sensitivity-specificity tradeoff is thus more robust because working points with sensitivity below the set threshold (application-dependent) are not valid, leaving only TCN points on the Pareto frontier. Even if the HDC can maximize specificity, reaching higher values compared to the TCN, the HDC's maximum-specificity points have lower sensitivity and higher delay.

In general, the sensitivity threshold is dictated by the desiderata of each particular scenario and is application-dependent, just like the preferred delay-specificity tradeoff point chosen on the Pareto curve for a specific use case. The detection results show that, depending on the desired sensitivity requirement, the TCN improves the Pareto frontier compared to the current SoA approach.

3.2.3.2 Deployment on a parallel MCU

Finally, the 8-bit TCN is deployed onto the multi-core MCU GAP8 (2.2) [33], [34], specialized for deep learning applications at the edge. Table 3.7 reports the deployment figures of merit, compared with the SoA HDC Ensemble. The model requires just 2.52 kB of model parameters storage and 164 kMAC = 328 k arithmetic operations, as detailed in 3.2.2.2 and 3.2.2.4. These values are 7.1× and 100× lower than the requirements of the HDC Ensemble SoA, respectively. The experimental values of computation latency and energy consumption were measured running the model on GAP8 at $V_{\mathrm{dd}} = 2.8\,\mathrm{V}$ and $f_{\mathrm{clk}} = 100\,\mathrm{MHz}$. Averages and standard deviations were taken over 20 repetitions of the model execution. The energy cost E was determined experimentally by measuring the consumed current $i(t)$ and integrating it over the model execution time:

$$E = V_{\mathrm{dd}} \int_0^T i(t)\mathrm{d}t \tag{3.7}$$

Table 3.7: Deployment metrics of the proposed TCN compared against the SoA HDC algorithm. Regarding operations, 1 MAC = 2 arithmetic operations.

Model:	**HDC Ensemble [154]**	**EEGNet-inspired TCN (this work)**	
Memory	17.8 kB	2.52 kB (0.14×)	
Arithm. op.	32.8 M	328 k (0.01×)	
Platform:	**Quentin**	**GAP8**	
		1-core	**8-core**
V_{dd}	0.52 V	2.8 V	
f_{clk}	187 MHz	100 MHz	
Cycles (k)	33100	568.1 ± 0.6 (0.017×)	146.4 ± 0.2 (0.005×)
Latency (ms)	177.0	5.681 ± 0.006 (0.032×)	1.464 ± 0.002 (0.009×)
Energy (µJ)	287.9	124.52 ± 0.27 (0.43×)	51.19 ± 0.13 (0.18×)

where $[0, T]$ is the time interval required for the execution, which was identified experimentally. These measurements yielded relative uncertainties of the order of 10^{-3}; this variability across repetitions is due to unpredictable cache effects of the GAP8 processor. This amount of variability is negligible for end-to-end use. Using 1 core, each inference requires, on average, just 5.68 ms of computation latency and 124.5 µJ of energy cost. Both these values are better than the SoA. Moreover, parallelizing the inference on all the 8 cores of GAP8 shortens the latency to 1.46 ms and decreases the energy consumption to 51.2 µJ. It is to remark that this 5.68 ms computation latency is negligible compared to the detection delay, which is of the order of seconds, as shown in Figure 3.9.

The obtained memory footprint, latency, and energy consumption prove that the solution successfully meets the requirements for implementation on resource-constrained devices.

Chapter 4

Classification: sEMG-based Hand Gesture Recognition

This chapter presents the three contributions of this thesis that belong to the task of multi-class, single-label classification. The first work (4.1) presents off-device TCNs for recognizing discrete hand gestures from the sEMG signal; these models prove accurate in cross-day scenarios. The second work (4.2) advances the cross-day sEMG recognition setup by presenting the TEMPONet TCN, which is fully deployed and profiled on a low-power parallel MCU. The third work (4.3) presents a heuristic based on online PCA for unsupervised adaptation to compensate for changes in arm posture. These contributions on embedded time-series ML/DL for multi-class classification are entirely devoted to the hand modeling from the sEMG; Chapter 5, which will address the finer task of multi-target regression, belongs entirely to the same field. The common purpose of these research contributions is the progress in non-invasive sEMG-driven intuitive HMIs, which must fulfill the resource constraints of embedded platforms to be implemented as wearable devices, as exposed in depth in Section 2.4.

4.1 Temporal Variability Analysis in sEMG Hand Grasp Recognition using Temporal Convolutional Networks

4.1.1 Overview

In this section, I address the challenge of the inter-session generalization of sEMG-based grasp recognition using TCNs. Exploiting the very challenging Non-Invasive Adaptive hand Prosthetics Database 6 (NinaPro DB6) [107], specifically utilized to investigate the algorithm adaptation over multiple sessions, this contribution tested several multi-session training set compositions to minimize the accuracy loss over non-training data. Moreover, this work evaluates the TCN-based approach against other well-established ML methods to explore the performance drop over multi-day testing sessions. Finally, these experiments evaluate the network quantization to allow the network deployment on platforms with a reduced memory budget (i.e., $< 512\,\text{kB}$). This contribution brings a twofold result:

- This work demonstrates that this approach can yield an inter-session classification accuracy of 49.4% on the NinaPro Database 6, improving by 7.6% the results achieved in literature and outperforming by 4.4% the results yielded by an RBF-SVM based on RMS feature, a widely used baseline for gesture recognition. This solution reaches a 4.8% accuracy drop on unseen data after 5 training sessions, demonstrating the feasibility of a reliable and robust controller.
- This work verifies that the implemented network can be distilled using data quantization and pruning, which allow deployment on resource-constrained real-time platforms, a fundamental requirement in next-generation design for wearable HMI. The presented model can fit a 512 kB memory, yet achieves 1.9% higher inter-session accuracy than the SVM, exhibiting 4× lower memory footprint than the SVM and 120× lower memory footprint compared to the complete network.

4.1.2 Materials & Methods

4.1.2.1 The NinaPro Database 6

The Non-Invasive Adaptive hand Prosthetics Database 6 (NinaPro DB6) [107] is a public sEMG dataset exploring the robustness of sEMG-based hand gesture recognition over time. A more in-depth exposition of the sEMG signal and sEMG-driven HMIs is provided in Section 2.4 The dataset comprises 10 able-bodied subjects (3 females, 7 males, average age of 27 ± 6 years). For each subject, data were recorded in 10 sessions (5 days, twice a day: morning and afternoon), each entailing 12 repetitions of 7 grasps. The grasps were chosen from the rehabilitation and robotics fields, selecting movements typical of the Activities of Daily Living. Each repetition lasts approximately 6 s, followed by 2 s of rest. The sEMG signals were acquired with 14 Delsys Trigno sEMG Wireless electrodes on the higher half of the forearm, sampling at 2 kHz. Figure 4.1 shows the sEMG trace of a grasping gesture, indicating the *rest*, *transient*, and *steady grasp* stages. It is noteworthy that, before and after the steady contraction, the gesture execution exhibits strong transient stages, characterized by signal amplitudes and RMS up to 5× greater than the steady signal.

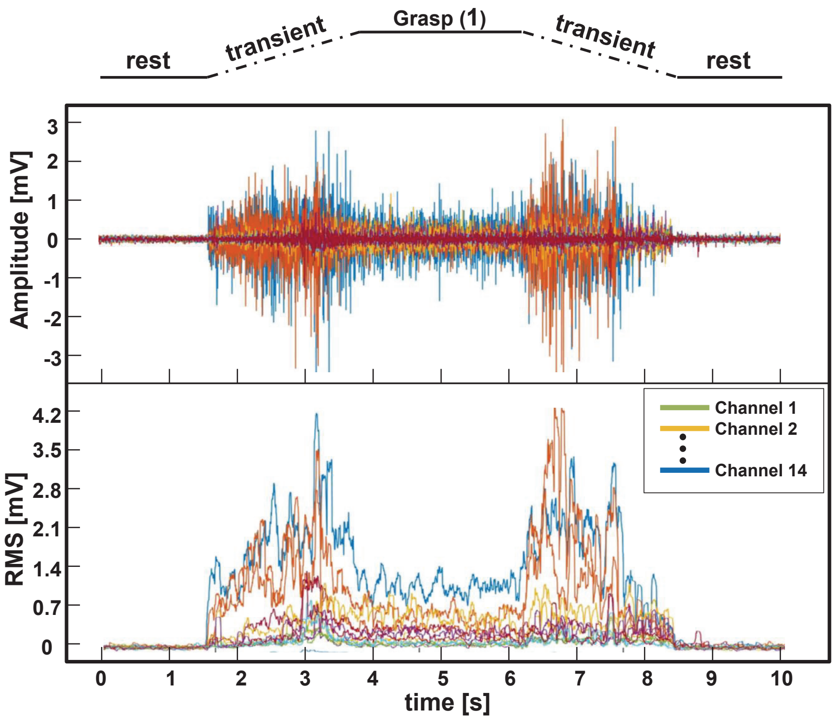

Figure 4.1: sEMG signal of the first grasp of NinaPro DB6, preceded and followed by the rest position. The beginning and the end of the grasp exhibit the strong transients typical of the dataset.

Remarkably, the NinaPro DB6 has been under-exploited in literature [107], [109], as it is made very challenging by the similar nature of the grasps, and by the aforementioned strong transient states caused by the impulsive movements. Results obtained by Palermo *et al.* [107], applying Random Forest (RF) on Waveform Length (WL), showed high inter-session (morning-to-afternoon) accuracy loss: training on morning data (for each day, for each subject) yields a 52.4% average accuracy on morning data, but a 25.4% average accuracy when validating on afternoon data. These results were improved by Cene *et al.* [109], who used Extreme Learning Machines (ELM), achieving 69.8% accuracy on the morning sessions used for training, and 41.8% accuracy in the afternoon. This work represents SoA on the NinaPro DB6 before the present contribution. The main limitation of the SoA is that the temporal variability is not properly addressed since the setup is limited to single-session training. The present contribution improves the SoA with a new setup, including multi-session training.

4.1.2.2 TCN model

This contribution treats the sEMG signal as a time series, employing a small TCN to classify fixed-size sEMG time windows. This approach significantly differs from previous

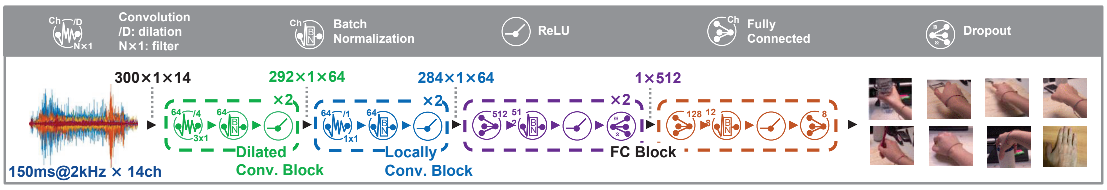

Figure 4.2: TCN architecture. Three main blocks are used: the *Dilated Convolutional Block* for temporal feature extraction, the *Local Convolutional Block* for local feature extraction, and the *FC Block* for the final classification.

works in sEMG-based gesture recognition, which are framed as single-sample classification [92] or image classification [98]. TCNs are introduced and explained in Section 2.1. Figure 4.2 displays the architecture of the proposed TCN:

1. the *Dilated Convolutional Block* is composed of 2 convolutional layers with $3\times1\times64$ filters, and dilation $d = 4$, to exploit the temporal sequence extracting information on the time-relationship of the samples;
2. the *Local Convolutional Block* is composed of 2 convolutional layers with $1\times1\times64$ filters, to capture the cross-information of the different channels;
3. finally, the *Fully Connected (FC) Block* is made of 3 FC layers, with dropout ($p_{\text{dropout}} = 0.5$) to help regularization [165], which flatten the input information to assign a label to the input sequence.

All layers have ReLU non-linearity as activation function and are equipped with BN to counter the internal covariate shift [117].

4.1.3 Experimental Results

Temporal variability and generalization to new sessions are addressed by developing a multi-session setup used to compare a set of conventional ML algorithms, namely LDA, RF, and Radial Basis Function-kernel Support Vector Machine (RBF-SVM), against DL models, namely the proposed TCN and its distilled versions. An incremental training protocol is adopted to analyze the ability to generalize to never-seen sessions, where 1 to a maximum of 5 sessions are used for training, and the remaining 5 for testing. In this sequential scenario, training sessions always precede testing sessions, preserving temporal coherence. An internal 2-fold cross-validation is employed; this 2-fold split is stratified, i.e., each fold contains an equal number of grasp repetitions from each training session. The 2-fold cross-validation is also necessary to evaluate the TCN algorithm on the same sessions used for training (i.e., without the temporal variability). Hence, the setup uses alternately one fold of the training sessions to train the model and the other

Table 4.1: Memory footprint and best intra- and inter-session accuracy of the proposed methods compared to the SVM baseline.

	Intra-/inter-session accuracy	**Memory footprint**
State of the Art [109]		
ELM	69.8%[1]/ 41.8%[1]	n.a.
Classical ML		
LDA	47.5% / 38.3%	negligible
RF	46.3% / 43.4%	negligible
RBF-SVM	51.4% / 45.0%	1.3 MB
RBF-SVM on steady	69.2% / 60.4%	1.3 MB
Proposed DL methods		
Full TCN	54.2% / 49.4%	38.8 MB (30×)
Full TCN on steady	71.3% / 65.0%	38.8 MB (30×)
Full TCN q. and prun.	53.0% / 49.3%	2.0 MB (1.5×)
Strided TCN	52.0% / 48.7%	6.0 MB (4.6×)
Strided TCN q. and prun.	49.5% / 46.9%	0.33 MB (0.25×)

[1]With transient removal by outlier rejection (4.1.3.2).

to test it. The proposed method is evaluated on two criteria: (1) the *intra-session* validation accuracy, calculated as the average accuracy on the fold not used for training (alternately), and (2) the *inter-session* validation accuracy, computed as the average accuracy on sessions 6-to-10, never included in the training.

4.1.3.1 Classical ML accuracy

First, multi-session training is applied on three well-established ML approaches: LDA, RF, and RBF-SVM, implemented with Python 3.5 and Scikit-learn 0.20.0 [166] using the modules `discriminant_analysis`, `ensemble`, and `svm`, respectively. All are fed the Root Mean Square (RMS) of the 14-channel sEMG signal, computed on 60 ms time windows. This approach was chosen as a baseline as it is a widely used setup. For all three classifiers, adding training sessions gradually improves the recognition of the last 5 never-seen sessions, with the 5-session training yielding the best accuracy, as reported in Table 4.1. This behavior is due to the regularizing effect of multi-session training: showing the classifier more heterogeneous data improves generalization. In particular, the RBF-SVM (trained with `C = 1` and `gamma = 'scale'`) yields the best results, shown in the left panel of Figure 4.3. With 5-session training, the SVM achieves an average intra-session validation accuracy of $(51.4 \pm 0.6)\%$ and an average inter-session validation accuracy of $(45.0 \pm 0.8)\%$, thus with a 6.4% accuracy drop on never-seen sessions. Sweeping from 1-session to 5-session training increases the intra-session validation accuracy by 1.1% and the inter-session validation accuracy by 8.2%, and reduces the inter-session

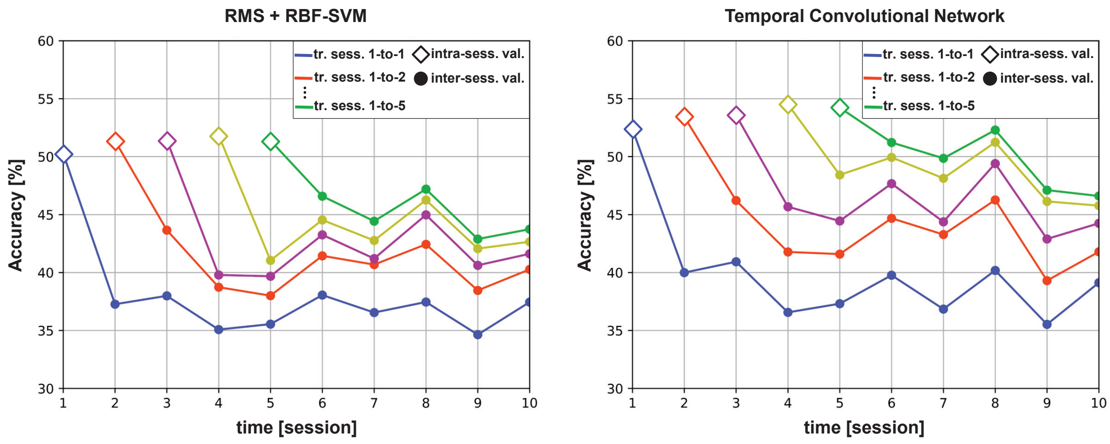

Figure 4.3: Classification accuracy of the baseline SVM and the proposed TCN on the different sessions of NinaPro DB6. The 5 multi-session training strategies incrementally improve the generalization to never-seen sessions, with better results for the TCN.

accuracy drop by 7.1%. This inter-session validation accuracy outperforms by 3.2% the SoA represented by [109] mentioned in 4.1.2.

4.1.3.2 TCN accuracy

The TCN-based approach is implemented using Python 3.5 and PyTorch 1.1 [65], [66]. The TCN is fed with 150 ms sliding (15 ms) time windows of the raw 14-channel sEMG signal, in sharp contrast with the classical ML methods. The TCN is trained with cross-entropy as a loss function and stochastic gradient descent for 20 epochs (minibatch size 64) with initial learning rate 0.001, divided by 10 after epochs 9 and 19. L_2-regularization is applied with PyTorch `weight_decay = 1e-4`, corresponding to $\lambda_{\mathrm{L2}} = 5 \cdot 10^{-5}$. Even though the training is done off-line, the proposed setup fully allows for online inference since the TCN neither relies on feature extraction nor the removal of sEMG transients (as instead popular in both inter-subject [167] and inter-session [90], [168], [169] studies, including the SoA on NinaPro DB6 [109]. The recognition accuracies obtained by adding training sessions are shown in the right panel of Figure 4.3, which provides a detailed comparison between the RBF-SVM and the TCN.

Multi-session training gradually increases the performance on the last 5 never-seen sessions, minimizing the inter-session accuracy drop, with the best results for the 5-session training strategy. This means that including more sessions enables the TCN to leverage more heterogeneous samples to learn a more robust representation. This behavior is the same as for the classical ML algorithms but with higher accuracy. Passing from 1-session to 5-session training improves the intra-session validation accuracy by 1.8% and the inter-session validation accuracy by 11.1%, and reduces the inter-session accuracy drop by 9.3%. The 5-session training achieves an average intra-session validation accuracy of $(54.2 \pm 0.6)\%$ (2.8% higher than the SVM) and an average inter-session validation accuracy of $(49.4 \pm 0.9)\%$ (4.4% higher than the SVM). The inter-session validation accuracy is better-than-SoA by 7.6% [109].

The lower-than-SoA intra-session accuracy is due to the fact that [109] (1) uses single-session training with feature extraction, prone to overfitting to the single sessions; (2) applies very aggressive signal filtering over 200 ms time windows, thus significantly smoothing the transients discussed in 4.1.2; and (3) embeds outlier removal into the algorithm, thus excluding the transients from the accuracy. For a fair comparison, the RBF-SVM and TCN are evaluated discarding the transients, obtaining results comparable to the SoA. Validated on the steady segments, the RBF-SVM yielded 69.2% intra-session accuracy (just 0.6% below SoA) and 60.4% inter-session accuracy, while the TCN achieved 71.3% intra-session accuracy (1.5% above SoA) and 65.0% inter-session accuracy.

The improvement achieved in inter-session accuracy is due to the successful regularization provided by multi-session training, as for the classical ML algorithms. Moreover, the higher results compared to classical ML corroborate the initial assumption that TCNs can achieve superior generalization thanks to their higher ability to process raw data and handle the variability of the sEMG signal between sessions.

4.1.3.3 TCN distillation

On top of evaluating the accuracy performance of the TCN-based approach, this subsection discusses the memory requirements of the proposed solution to understand how it can be deployed on a resource-constrained platform. Three new networks are distilled from the initial one, applying (1) a stride factor s in the first two convolutional layers ($s_1 = 2$ in the first and $s_2 = 4$ in the second), (2) 16-bit quantization for the convolutional layers and 8-bit quantization for the FC layers, and (3) the pruning of the network weights.

Table 4.1 reports the memory occupancy and the accuracy of the different configurations of the network, compared to the baseline SVM. Remarkably, introducing the strides, quantizing, and pruning causes an accuracy loss of only 4.7% intra-session and 2.5% inter-session. This lowest-area configuration requires 120× less memory than the initial configuration and 4× lower memory footprint compared to the SVM, yet demonstrates an inter-session accuracy higher than the SVM. In addition, neural networks can benefit from a far better parallelization than SVMs, leading to lower delay in the recognition.

4.2 Robust Real-Time Embedded EMG Recognition Framework Using Temporal Convolutional Networks on a Multicore IoT Processor

4.2.1 Overview

In this section, I address the challenge of variability robustness of sEMG-based hand gesture recognition, and I present a real-time embedded platform for robust sEMG-based gesture recognition. The major contributions are:

- TEMPONet, a novel EMG classification algorithm based on a TCN, tested on the benchmark sEMG dataset NinaPro DB6 (detailed in 3.1.3.3);
- a complete embedded platform for EMG acquisition and processing. The system is based on the combination of a commercial Analog Front End for biopotential acquisition with GAP8 (2.2), a multi-core low-power IoT processor;
- a 20-session dataset, collected with the proposed custom platform, which allows the validation of the algorithm and profiling of a quantized version of the TCN suitable for the deployment on a resource-constrained platform.

The performance of TEMPONet is tested on the NinaPro DB6 dataset [107] achieving 65.2% inter-session accuracy on steady signals and 49.6% inter-session on the full dataset – 7.8% better than the current SoA [109]. Moreover, after the system design, the same TEMPONet topology is tested on a new dataset introduced in this work, comprising 20 sessions on 3 subjects. On this dataset, the setup achieves 93.7% inter-session accuracy. The 20-session dataset is collected using the same platform on which the recognition algorithm is deployed. Therefore, the dataset is representative of the real-world of data that an embedded setup can gather. The results show that the accuracy drop on entirely unseen sessions can be reduced to 6.6% on NinaPro DB6 and 3.4% on the 20-session dataset, surpassing the current SoA. Finally, a full quantization of TEMPONet is performed, dropping the data representation of the weights and the feature maps from 32-bit floating point to 8-bit integer, thus reducing the network memory footprint by 4×. To leverage the 8-core architecture of the GAP8 processor and parallelize the execution of the algorithm, the deployment exploits highly optimized neural network libraries [170]. The quantized TCN can be executed in real-time on the GAP8 (2.2) chip (a full inference takes less than 13 ms and consumes 0.9 mJ), but still achieves 93.3% inter-session accuracy on the 20-session dataset and 61.0% on the NinaPro DB6 while providing up to 54 h of battery life, showing a computational efficiency of 10× compared

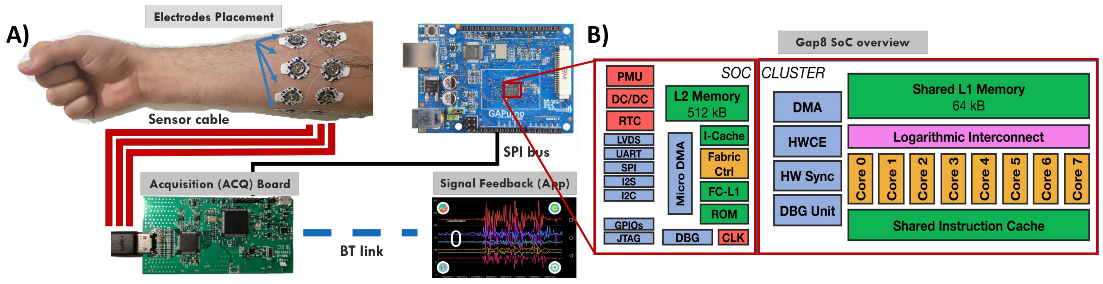

Figure 4.4: A) Hardware diagram of the proposed system: the sEMG sensors on the forearm are connected to the AFE, which sends the data via SPI to the GAP8 processing platform; a Bluetooth link allows the streaming of the data and the classification to an external gateway. B) Detailed block diagram of the GAP8 processor (2.2).

to SoA systems for sEMG processing suitable for convolutional network deployment, such as [171] and [172].

4.2.2 Materials & Methods

4.2.2.1 Acquisition & processing platform

The sEMG signal acquisition is based on an 8 channel commercial Analog Front End (AFE), an ADS1298 [173] connected to the GAP8 (2.2) breakout board. ADS1298 is mainly used in acquisition system design for EMG, EEG, and ECG signals, and it is considered the *de facto* standard for such applications. It allows simultaneous sampling of up to 8 bipolar channels with 24-bit resolution, reaching 32 ksps. Each channel has a programmable Gain Amplifier with a gain that ranges from 1 to 12. In this application, it drives 8 fully differential channels at sampling rate 4 kHz connected to an array of passive gel-based sEMG electrodes, placed in a ring configuration around the forearm. The block diagram of the GAP8 architecture (2.2) is reported in Figure 4.4. GAP8 has two main functional blocks: a single tiny RISC-V core, namely the Fabric Controller (FC), and a parallel set of 8 RISC-V core, i.e., the computational cluster. The FC controls the SoC and the peripherals and can be viewed as a simple MCU. The 8-core cluster is used for vectorized and parallel computationally-intensive tasks such as embedded artificial intelligence [174].

GAP8 is not equipped with an FPU. Hence, algorithms need to be executed using fixed-point arithmetic. The internal memory of GAP8 is divided into two layers: L1 memory and L2 memory. L2 memory is 512 kB in size and accessible by all cores. L1 memory is split into two parts: a 16 kB memory for the fabric controller and a 64 kB shared memory for the cluster cores. There is also a third level, i.e., L3, externally connected via quad-SPI or a HyperBus interface. The GAP8 processor also includes an internal programmable DC/DC converter which provides power supply to the fabric controller and cluster (0.9 V to 1.3 V; 0.8 V for retentive sleep mode). As shown in Figure 4.4, this setup is developed for measurement and characterization using development boards; nevertheless, by virtue of the BGA packages of the ADC and GAP8, the whole system can be integrated into a single PCB with 30 mm$times$20 mm form factor suitable for wearable applications.

All firmware was written in C and runs on the low-power GAP8 processor. The software development relied on the GAP8 Software Development Kit (SDK) [175], which embeds all the APIs to access the GAP8 hardware features such as DMA engines, hardware timers, and I/O. The GAP8 SDK also includes a customized version of the RISC-V GCC compiler with support for the GAP8 ISA extensions used to accelerate the inference of DNNs [170]. As shown in Figure 4.4, GAP8 is connected to the ADS1298 AFE via a 20 MHz SPI connection (GAP8 acts as master) in parallel to an interrupt wire (`#DRDY`) connected to a GAP8 digital pin. Once a new sample is ready, the AFE asserts the `#DRDY` signal with a pulse. The `#DRDY` pulse wakes up an interrupt routine on the GAP8 FC, which starts acquiring the SPI data using the embedded I/O µDMA. Data loaded via SPI is stored in the GAP8 L2 memory as 24-bit signed fixed-point numbers, with the least significant bit representing a value of $V_{\mathrm{ref}}/(2^{23}–1)$. Acquired sEMG samples are then used as input of the TEMPONet TCN, whose embedded implementation is described in 4.2.2.3 and profiled in 4.2.3.

4.2.2.2 TEMPONet TCN architecture

Unlike previous studies in sEMG-based gesture recognition, which are formulated as single-sample recognition [92] or image recognition [98] and mostly rely on extracted features, this contribution addresses the sEMG signal as a time series, using a small TCN-based architecture to assign labels to 150 ms raw sEMG time windows. An in-depth presentation to TCNs is provided in Section 2.1. Figure 4.5 portrays an example of TCN convolutions acting on a raw signal without any preliminary feature extraction; this work is built on this principle. This section presents the novel TCN-based topology, the Temporal Embedded Muscular Processing Online Network (TEMPONet), also displayed in Figure 4.6. TEMPONet stacks 3 Convolutional Blocks, each composed by:

- 2 temporal convolutional layers with filter size 3×1, variable dilation, and full padding;
- 1 convolutional layer with filter size 5×1 and variable stride and padding, followed by an Average Pooling (AvgPool) with kernel 2×1.

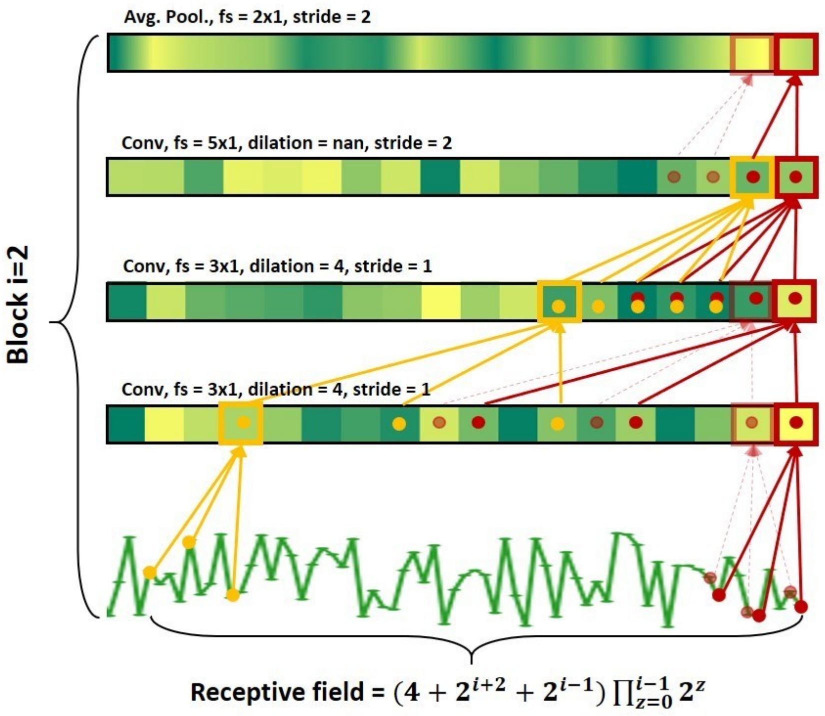
Avg. Pool., fs = 2x1, stride = 2
Conv, fs = 5x1, dilation = nan, stride = 2
Conv, fs = 3x1, dilation = 4, stride = 1
Conv, fs = 3x1, dilation = 4, stride = 1
Block i=2
Receptive field = $(4 + 2^{i+2} + 2^{i-1}) \prod_{z=0}^{i-1} 2^z$

Figure 4.5: Structure and functioning of TEMPONet's 2nd Convolutional Block: 2 dilated convolutions ($d = 4$), 1 strided convolution ($s = 2$), and average pooling. The input of the block is the temporal sequence computed by the 1st Convolutional Block.

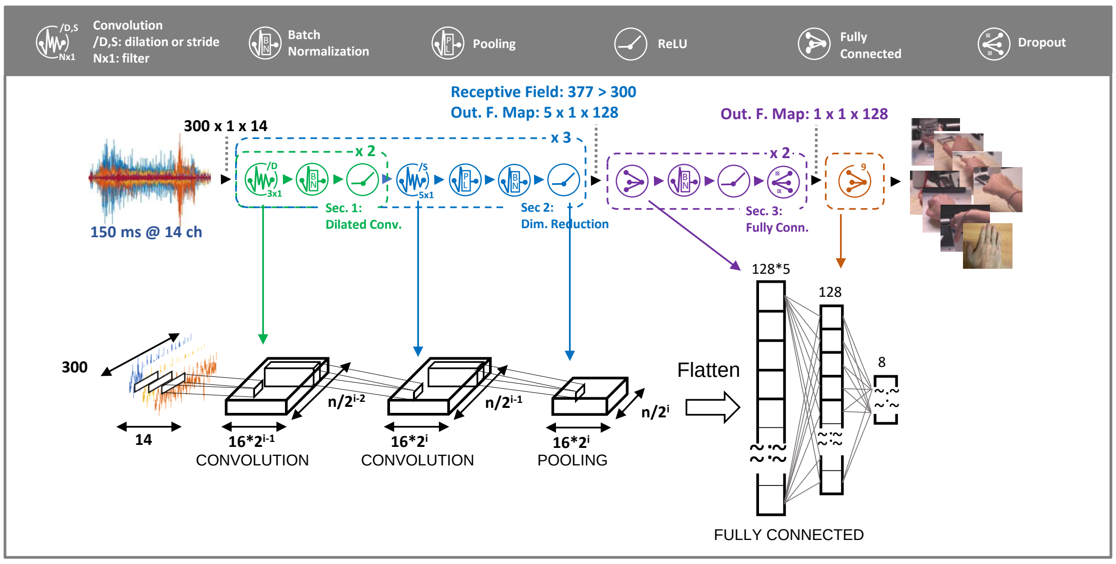

Figure 4.6: Processing diagram of the proposed algorithm. In the TEMPONet TCN architecture, the three blocks (each one composed of 2 convolutional and one pooling layer) are used to extract temporal features, followed by two fully connected layers that perform the final classification.

The 3 blocks are characterized by dilation $d = 2, 4, 8$, respectively, and stride $s = 1, 2, 4$, respectively. The strided convolution of the 1^{st}, 2^{nd}, and 3^{rd} block raises the number of channels to 32, 64, and 128 respectively, whereas each AvgPool halves the sequence length immediately after. As an example, Convolutional Block 2 is represented in Figure 4.5. The Convolutional Blocks are followed by 2 Fully Connected (FC) layers with dropout (to help regularization [165]) and a SoftMax operation. The FC layers flatten the input information to compute the final label assigned to the sequence. All layers have ReLU non-linearity as activation function and are equipped with BN to counter the internal covariate shift [117].

The two main characteristics of this network, namely block composition and 1D dilated convolutional layers, are inspired by the novel developments in the deep learning field. The division into blocks of several layers where the number of channels and size of the activation tensors is kept constant is typical of many modern networks [176]–[179]. It enables building a network where the temporal dimension is consumed "slowly," thus enabling a deeper network with more powerful processing of the raw information in the time series. On the other hand, dilated layers allow to gradually increase the receptive field of each layer (2.1). Dilation factors are chosen so that the receptive field at the network's end covers an entire time window. The modular nature of the network structure would allow stacking further blocks so as to process signals on different timing windows; for the sEMG-based gesture recognition, this is of particular interest since the time-window width could change based on the target application.

The network topology is designed so that the convolutional block increases the receptive field and reduces the width of the signal (i.e., the input time window "visible" from a given neuron in the layer). A larger input window can then be analyzed by stacking more blocks instead of increasing the filter sizes, i.e., by making the network deeper instead of wider, thus limiting the increase in the number of parameters. Combining these insights, the TEMPONet 3-blocks configuration presented in this paper can process a 150 ms input window using only 460 k parameters, which is well suited for implementing the setup described in the following parts of this section.

4.2.2.3 TEMPONet embedded deployment

This section describes the procedure to distill the TEMPONet algorithm for the embedded platform for sEMG acquisition and classification.

Quantization As mentioned in 4.2.2.1, the target execution platform GAP8 has limited memory capacity and no support for floating-point data. Therefore, to enable deployment of the trained TEMPONet on an embedded platform, quantization (2.3.1) is required to reduce the net to only 8-bit integer parameters and feature map tensors. The pre-trained TCNs are fine-tuned after replacing ReLU activation functions with step functions using the PACT methodology [67]; weights are also quantized using a similar function. Quantization is performed layer-wise. The 8-bit representations of feature maps $\mathbf{y}$ and weights $\mathbf{W}$ are given by (respectively)

$$\widehat{\mathbf{y}} = \left\lfloor \frac{\mathrm{clip}_{[0,\alpha_{\mathbf{y}})}(\mathbf{y})}{\varepsilon_{\mathbf{y}}} \right\rfloor, \qquad \text{with} \quad \varepsilon_{\mathbf{y}} = \frac{\alpha_{\mathbf{y}}}{256}; \tag{4.1}$$

$$\widehat{\mathbf{W}} = \left\lfloor \frac{\mathrm{clip}_{[\alpha_{\mathbf{W}},\beta_{\mathbf{W}})}(\mathbf{W})}{\varepsilon_{\mathbf{W}}} \right\rfloor, \qquad \text{with} \quad \varepsilon_{\mathbf{W}} = \frac{\beta_{\mathbf{W}} - \alpha_{\mathbf{W}}}{256}. \tag{4.2}$$

The quantization procedure operates as follows, starting from a pre-trained full-precision network: first, the quantization parameters $\alpha_{\mathbf{W}}$ and $\beta_{\mathbf{W}}$ are initialized with the minimum and maximum values of $\mathbf{W}$, respectively, while $\alpha_{\mathbf{y}}$ parameters are initialized by registering minimum and maximum values of $\mathbf{y}$ over a run on the training set. All parameters (including $\mathbf{W}$, $\alpha_{\mathbf{W}}, \beta_{\mathbf{W}}, \alpha_{\mathbf{y}}$) are then fine-tuned via back-propagation using the Adam optimizer. The learning rate is set to a small value (10^{-6}) for both datasets, and the training is stopped after 30 epochs or earlier if convergence is achieved (i.e., if the difference in loss between two epochs is < 0.05). The quantized network can be deployed on GAP8 by directly using the `int8` weights $\widehat{\mathbf{W}}$ and implementing Equation 4.1 as a set of comparisons against thresholds [180]. Apart from fitting in the GAP8 L2 512 kB memory constraint (the 8-bit TEMPONet has a footprint of 460 kB), quantization also removes all floating-point multiplications, reducing both the time and energy per classification as GAP8 has no floating-point unit and would emulate these operations in software.

Optimized execution The GAP8 processor receives and accumulates the data to fill an internal 150 ms $\times$ 8 channel (i.e., $\sim$ 10 kB) buffer in L2 and then starts the classification. Meanwhile, a second buffer, also located in L2, receives the data in real-time from the AFE in a double-buffering procedure. Data is fed to the network at 2 kHz sampling rate and using `int32` representation only for the input data (as the ADC resolution is 24 bits – 4.2.2.1). The implementation of TEMPONet on GAP8 is based on the dedicated PULP-NN libraries [170] for optimized ad-hoc convolutional kernels deployment. PULP-NN uses all the cores available in the GAP8 cluster, their SIMD extensions, and their bit-manipulation instructions to obtain the best speed-up and energy-efficiency from the chip. As PULP-NN functions work on data in the 64 kB L1 scratchpad, it is necessary to move weights and feature maps between the 512 kB L2 memory and the L1 scratchpad. This process is performed by using the automated tool DORy (2.3.2) [41], [42] to divide the data tensors in each layer in tiles that fit the L1, and

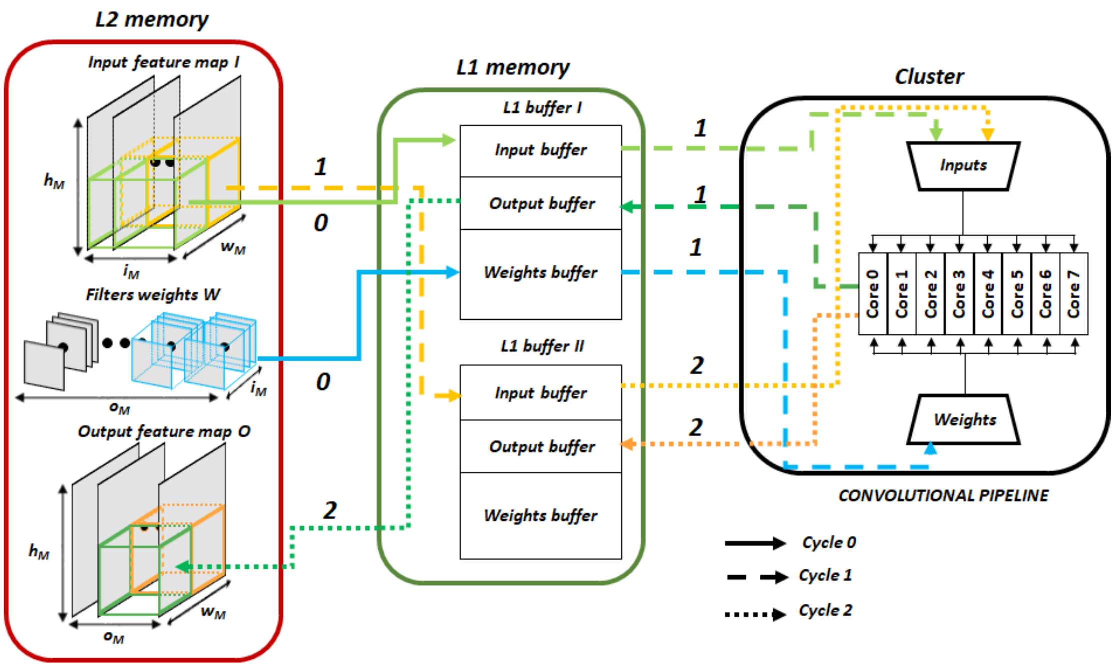

Figure 4.7: TEMPONet flow. Left: the DMA manages L2-L1 communication using double-buffering. Right: the cluster executes PULP-NN on a tile stored in one of the L1 buffers.

to insert appropriate DMA calls to realize a double buffering scheme (separate from the one on ADC data) so that data movement is always overlapped with computation. The DMA calls are asynchronous and non-blocking, allowing new activations and weights to be imported while the previous calculation is ongoing. Figure 4.7 illustrates this flow by highlighting the L2-L1 memory traffic, managed by the DMA, and the cluster execution of PULP-NN.

4.2.3 Experimental Results

4.2.3.1 Experimental setup

The TEMPONet TCN is implemented using Python 3.5 and the specialized DL development PyTorch 1.1 framework [65], [66]. The TCN is fed with 150 ms time windows (slide 15 ms) of the raw 14-channel and 8-channel sEMG signal, for the NinaPro DB6 and the 20-session dataset respectively. To analyze the accuracy drop in multi-session classification, the setup uses an incremental training protocol that sweeps the training data from 1 to a maximum of half dataset sessions (i.e., 5 for NinaPro DB6 and 10 for the 20-session dataset), using the remaining half for testing. The training sessions always precede the testing ones in a sequential scenario to maintain temporal coherence among sessions. Regarding the amount of data used for the training, the strategy adopts an internal 2-fold stratified (i.e., with an equal number of gesture repetitions for each fold) cross-validation to evaluate the algorithm also on the same sessions used for training (i.e., without the temporal variability). TCN training uses cross-entropy as a loss function and stochastic gradient descent for 20 epochs (minibatch size 64) with L_2 regularization (PyTorch's `weight_decay` $= 10^{-4}$). The initial learning rate is set to 0.001 for the NinaPro DB6 dataset and 0.01 for the 20-session dataset; in both cases, it is divided by 10 at epoch 9 and 19. It is worth observing that these training settings are very similar to the ones used in the previous contribution on NinaPro DB6, presented in Section 4.1; these settings proved heuristically effective on the new 20-session dataset as well. Furthermore, TCN topology is compared against an RBF-SVM applied on the RMS of the sEMG signal, computed on 60 ms time windows. As in 4.1, this setup is chosen as a baseline since it is a widely used approach [123] and it allows to show how the algorithm performs against a well-established classification scheme. The SVM is implemented using the Scikit-learn framework (version 0.20.0) [166], and the SVM parameters are set to `C = 1` and `gamma = 'scale'`.

Two figures of merit are introduced to evaluate the method, based on the protocol: (1) the *intra-session* validation accuracy, computed as the average accuracy on the fold not used for training (alternately), and (2) the *inter-session* validation accuracy, calculated as the average accuracy on sessions 6-to-10 for the NinaPro DB6 and 11-to-20 for the 20-session dataset, which are never used for training. Network training is always performed offline, hence on *steady* segments, removing contraction transients. Steady segments are obtained by discarding the first and the last 1.5 s of each gesture for NinaPro and 300 ms for the dataset introduced in this work, thus focusing the classification only on steady signal portions.

To evaluate the inference performance, TEMPONet is run on the Ninapro DB6, comparing the results of the multi-session testing described in [109] that represents the previous SoA inter-session accuracy for NinaPro DB6. The obtained average accuracy is 49.6% against 41.8% reported in [109]. In this test, the accuracy is evaluated on the complete gestures, including transients). Figure 4.10 displays the results of the RBF-SVM and the TCN on the NinaPro DB6 dataset, evaluated on the full validation set. The algorithm comparison shows that the TCN performs 4.3% better than the SVM. All accuracy results are reported as mean (i.e., average over subjects, training folds, and validation sessions) or as mean ± Standard Error.

It is noteworthy that since most of the classification errors are located in the transients, the design of an end-to-end gesture controller usually requires removing gesture transients. This procedure is well-established and common to both inter-session [90],

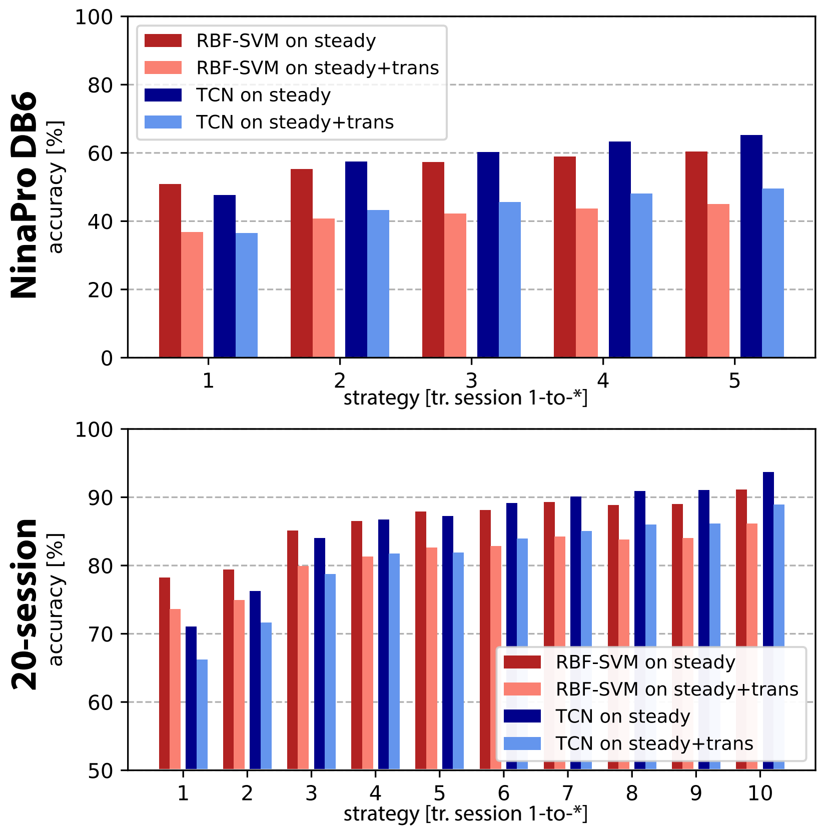

Figure 4.8: Average inter-session accuracy obtained on steady signals (removing transients, in full color) and on full signals (including transients, in light color) for both datasets and both algorithms.

[168], [169] and inter-subject [167] studies; this procedure is also sufficient for the purpose of a steady gesture controller design. It can be done using techniques such as threshold comparison, Dynamic Time Warping, or Hidden Markov Models. For this reason, the following paragraphs present the accuracy results also with data purged of transients. To provide better insight into how the accuracy varies when removing transients, Figure 4.8 compares the average inter-session accuracy achieved testing only on steady signals and on full ones for both datasets and considering both RBF-SVM and TEMPONet TCN approaches. Similarly to what happens with steady signals, the advantage of TEMPONet in terms of accuracy on full grows proportionally to the number of sessions involved in the training. The accuracy drop on full compared to steady is substantially similar between RBF-SVMs and TCNs for both datasets.

4.2.3.2 Accuracy on NinaPro DB6 (steady)

On the NinaPro DB6, both the SVM and the TCN yield the best recognition accuracy when trained with a higher number of sessions, namely 1-to-5 (i.e., the first half of the dataset sessions). Recognition accuracy over time is plotted in Figure 4.11. The

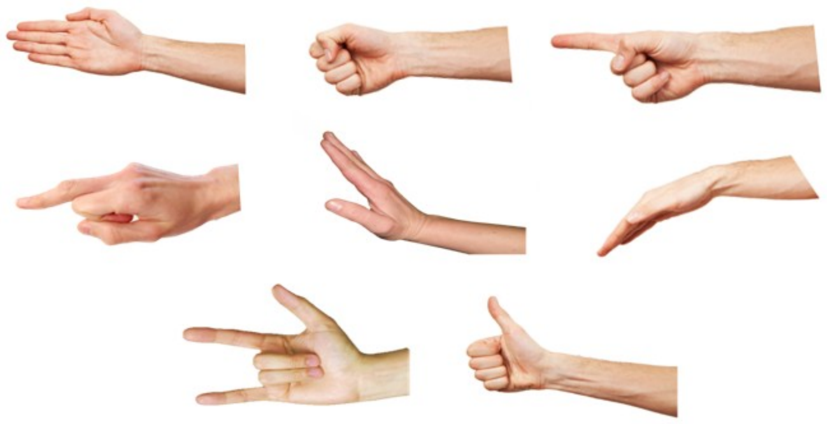

Figure 4.9: Hand gestures used during experimentation, including finger and wrist contractions.

SVM trained on sessions 1-to-5 reaches an average intra-session validation accuracy of $(69.2 \pm 0.7)\%$, and an average inter-session validation accuracy on sessions 6-to-10 of $(60.4 \pm 0.9)\%$, resulting in a drop of 8.8%. Compared to training on only session 1, the 5-session training maintains the same intra-session accuracy (+0.4%) but increases the inter-session accuracy by 9.5%. The TEMPONet TCN trained on sessions 1-to-5 reaches a similar intra-session validation accuracy of $(71.8 \pm 0.7)\%$ (2.6% higher than the SVM); it also increases the inter-session validation accuracy by 4.8% compared to the SVM ($(65.2 \pm 1.0)\%$), with a resulting drop of 6.6%. Compared to training on only session 1, the 5-session training increases the intra-session accuracy by 5.5%) and significantly increases the inter-session accuracy by 17.5%. Remarkably, increasing the amount of training data is crucial for TEMPONet, which strongly increases its performance. These results confirm the initial assumption that TCNs are more efficient in extracting information directly from raw data and removing part of the noise due to temporal variability, thus achieving better generalization over time.

4.2.3.3 Accuracy on the 20-Session Dataset (steady)

The new dataset was acquired for 10 days and involved 3 subjects (all male, average age of 29 ± 3 years). Each day includes 2 sessions, taking place in the morning and afternoon, for a total of 20 sessions for the complete experimentation. A single session has an approximate duration of 1.5 minutes and includes 8 hand gestures and rest, as shown in Figure 4.9. Each gesture is repeated 6 times with a contraction time of approximately 3 s. To ease the labeling process, 3 s of rest are interleaved between contractions of the same gestures, and up to 5 s of rest are interleaved between different gestures.

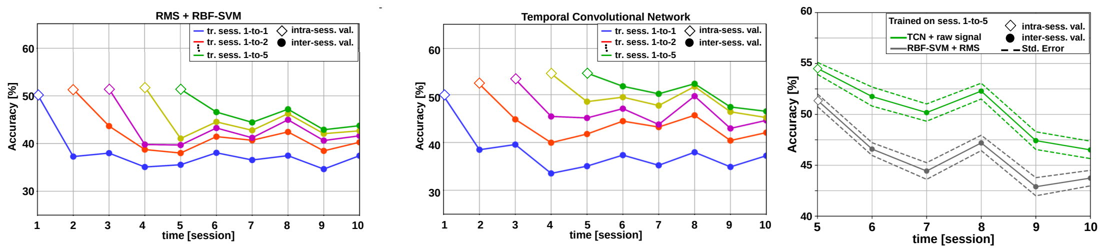

Figure 4.10: Left and Center: classification accuracy of RMS + RBF-SVM and TEMPONet, using the incremental training framework on NinaPro DB6. Right: classification accuracy of RMS + RBF-SVM and TEMPONet, after training on sessions 1-to-5 of NinaPro DB6. All validations are done on steady states + transient states.

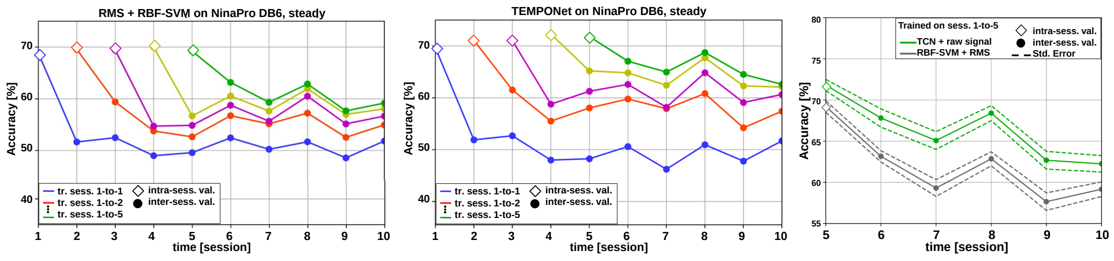

Figure 4.11: Left and Center: classification accuracy of the baseline RMS + RBF-SVM and TEMPONet, reached on the different sessions of NinaPro DB6 after transient removal, using the incremental multi-session training framework. Adding training sessions improves accuracy in the never-seen sessions, with better results for the TEMPONet TCN. Right: classification accuracy of RMS + RBF-SVM and TEMPONet, after training on sessions 1-to-5 of NinaPro DB6. All validations are done on steady states.

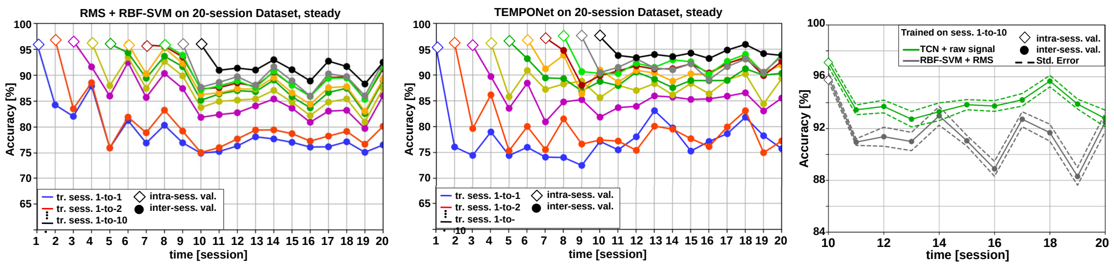

Figure 4.12: Left and Center: classification accuracy of the baseline RMS + RBF-SVM and TEMPONet, reached on the different sessions of the new 20-session Dataset after transient removal, using the incremental multi-session training framework. Adding training sessions improves accuracy in the never-seen sessions, with better results for the TEMPONet TCN. Right: classification accuracy of RMS + RBF-SVM and TEMPONet, after training on sessions 1-to-10 of the 20-session Dataset. All validations are done on steady states.

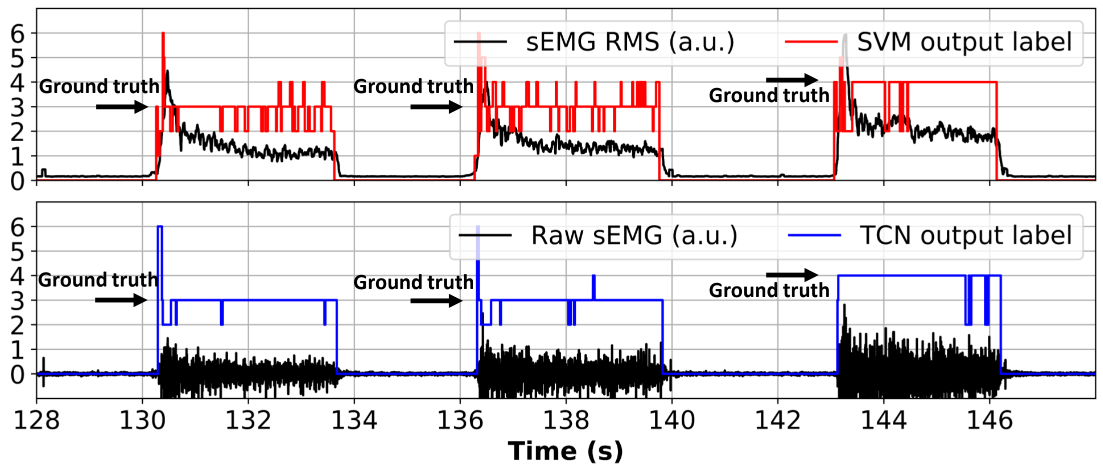

Figure 4.13: Comparison of the real-time inter-session classification of the RBF-SVM and TEMPONet on the 20-session dataset.

The SVM and the TEMPONet TCN were tested again on this new 20-session dataset. The same topology and training parameters are maintained, except for the learning rate (4.2.3). The recognition accuracy when using the incremental training protocol over time is plotted in Figure 4.12. Again, both the SVM and the TEMPONet TCN reach the best average intra-/inter-session validation accuracy when trained with the maximum number of training sessions (1-to-10). The SVM reaches $(96.0 \pm 0.3)\%$ and $(91.1 \pm 0.6)\%$, intra-session and inter-session accuracy (on sessions 11-to-20), respectively, and 4.9% drop. The TEMPONet increases this performance to $(97.1 \pm 0.3)\%$ intra-session accuracy, $(93.7 \pm 0.5)\%$ inter-session accuracy, and only 3.4% accuracy-drop. Compared to training on only session 1, the SVM maintains the same intra-session accuracy (-0.2%), while the TEMPONet strongly increases its performance of 9.4%. Remarkably, both methods enhanced with more training sessions show a sharp performance gain of the inter-session accuracy, namely 12.9% for the SVM and 22.7% for the TCN. Similarly to NinaPro DB6, the effect of multi-session training on the SVM and the TCN is similar but with higher gains for the TCN. The fact that the TEMPONet TCN also outperforms the SVM on the new 20-session dataset further validates the initial hypothesis that TCNs can attain better generalization by their higher ability to process raw data and handle the inter-session sEMG variability noise.

Furthermore, a more explicit comparison between the recognition accuracy of the SVM and TEMPONet is shown in Figure 4.13, which displays the output labels of the two classifiers in the real-time inter-session setup on the 20-session dataset (subject 1, training sessions 1-to-10, validation session 20). This visual inspection highlights that the output sequence returned by TEMPONet is more accurate and more stable (i.e., smooth) in inter-session validation than the output of the SVM. The smoothness of the TEMPONet TCN classification is due to the fact that, for each inference, TEMPONet leverages 150 ms of signal history, rich enough to enhance stability and avoid erratic oscillations as the ones exhibited by the SVM.

Finally, it is possible to notice that the classification accuracy reached on the 20-session dataset (all $> 90\%$) is consistently much higher than on NinaPro DB6 (all $<$ 75%), even in presence of a similar number of classes (8 for NinaPro DB6 vs. 9 for the 20-session dataset) and sEMG sensors (14 for NinaPro DB6 vs. 8 for the 20-session dataset). The cause is that hand movements in NinaPro DB6 are all grasps, thus much less diverse and discernible than the hand gestures of the 20-session dataset.

4.2.3.4 Embedded deployment performance

Regarding the embedded implementation, the input sEMG signals are preprocessed digitally before executing the TEMPONet TCN. This process includes a 10-tap notch filter to remove PLI interference and a 15-tap band-pass filter between 2 Hz and 1 kHz to cancel the DC drift and high-frequency components. The signal is then downsampled to 2 kHz to match the sampling rate of the NinaPro DB6 dataset. The execution time of these steps is negligible ($< 100\,\mu$s) and does not affect the real-time performance of the classifier. The processing chain later continues with the execution of TEMPONet as described in Figure 4.7.

To fairly evaluate the accuracy of the quantized version of TEMPONet, distillation to `int8` also involved the RBF-SVM support vectors. This was performed offline by evaluating the mean μ and standard deviation σ of the support vectors and applying Equation 4.2, setting

$$\alpha_{\mathbf{W}} = \mu - 5\sigma \tag{4.3}$$

$$\beta_{\mathbf{W}} = \mu + 5\sigma \tag{4.4}$$

(empirically, different settings for $\alpha_{\mathbf{W}}$ and $\beta_{\mathbf{W}}$ resulted in larger accuracy drops. Table 4.2 reports the memory occupancy and the accuracy of the full-precision and 8-bit TEMPONet, compared to the SVM baselines. Remarkably, the accuracy drop after quantization decreases, given the quantization's regularizing effect. On NinaPro DB6, quantization leads to an accuracy loss of 7.3% intra-session and 4.2% inter-session, still above the full-precision RBF-SVM baseline. On the 20-session dataset, quantization causes an accuracy loss of just 0.5% intra-session and only 0.4% inter-session, again above the full-precision SVM, but with a 1.5× lower memory footprint. On the other hand, the quantization of the support vectors, which is still necessary to deploy SVMs on the GAP8-based processing platform (512 kB memory constraint), results in a $\sim 15\%$ inter-session accuracy loss for both the datasets.

Table 4.2: Memory footprint and best intra- and inter-session accuracy of the baseline RMS + RBF-SVM, full-precision TEMPONet and 8-bit quantized TEMPONet.

	Memory footprint	**Intra-session accuracy (%)**	**Inter-session accuracy (%)**
NinaPro DB6			
RMS + RBF-SVM (`float32`)	1.3 MB	69.2	60.4
RMS + RBF-SVM (8-bit)	332 kB	50.7	44.7
TEMPONet (`float32`)	1.8 MB	71.8	65.2
TEMPONet (8-bit)	460 kB	64.5	61.0
20-Session Dataset			
RMS + RBF-SVM (`float32`)	670 kB	96.0	91.1
RMS + RBF-SVM (8-bit)	168 kB	95.8	78.6
TEMPONet (`float32`)	1.8 MB	97.1	93.7
TEMPONet (8-bit)	460 kB	96.6	93.3

Table 4.3: Inference latency and energy consumption of TEMPONet executed on GAP8 in the most efficient voltage-frequency configuration, namely 1.0 V and 170 MHz.

	Inference latency (ms)	**Inference energy (mJ)**	**MAC/cycle**
Dilated convolutions	5.40	0.38	9.54
Non-dilated convolutions	5.86	0.41	6.95
Averag poolings	0.16	0.01	n.a.
Fully connected	1.42	0.10	4.10
Whole net	12.84	0.90	7.73

Table 4.3 highlights the performance of the TCN network in terms of inference time, energy, and MAC/cycle, experimentally profiled from the execution of the net on the GAP8 SoC targeting the most efficient voltage-frequency configuration to save energy, namely $V_{\mathrm{dd}} = 1.0\,\mathrm{V}$ and $f_{\mathrm{clk}} = 170\,\mathrm{MHz}$. Also, these metrics are broken down for the different layer types involved, namely dilated and non-dilated convolutions, AvgPool, and Fully Connected layers. The last column of Table 4.3, the mean MAC/cycle, is a key indicator of computational efficiency. Dilated Convolutions are not only algorithmically effective, but they also achieve the highest level of efficiency: 42% of the execution time is spent to run 53% of the overall network operations. Overall, exploiting the 8 cores of GAP8, the network reaches a mean MAC/cycle of 7.73. Therefore, TEMPONet can classify a time window in 12.8 ms, consuming 0.90 mJ. The real-time constraint is given by the 15 ms of the sliding window and is therefore well-met by the embedded application, as shown in Figure 4.14. Moreover, Figure 4.14 highlights that the windowing scheme (same for off-line training and real-time inference) and computation comply with the consensus real-time requirement, which is represented by the upper limit of 300 ms [103]. Regarding classification energy, each time window classification

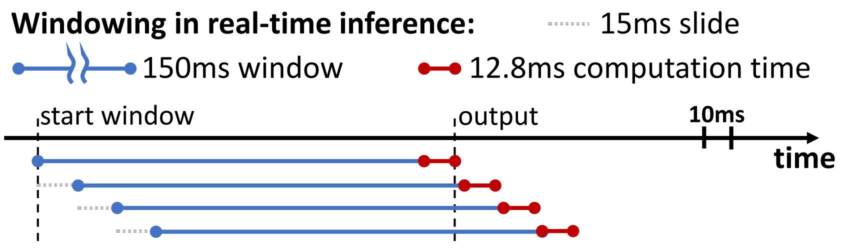

Figure 4.14: The windowing scheme and inference time. The system fulfills the real-time requirement for sEMG-driven hand HMIs, which is an upper limit of 300 ms [103].

costs 0.90 mJ per inference. Between two adjacent inferences, the GAP8 SoC is only collecting data (and not processing it) for 2.2 ms. During this phase, it is possible to idle the 8 core cluster using its embedded hardware synchronization unit [181], which enables fully state-retentive clock gating and wakeup in a few nanoseconds. The power consumption in this phase is limited to the $\sim$ 10 mW consumed by the SoC to collect data from the sensor. Overall, a 15 ms window costs 0.90 mJ, yielding an average power of 60 mW. Using a small 1000 mAh battery, the sEMG gesture classification system can run continuously for $\sim 13 \cdot 10^6$ classifications, i.e., for a lifetime of $\sim$ 54 h.

To measure the computational impact of dilated convolutions as opposed to conventional ones, it is possible to project the measured results over a modified version of TEMPONet where dilation factors are removed. Still, the dimension of the receptive field is kept constant, covering the same time window as TEMPONet. To do so, the filter sizes are increased to 5, 9, and 17 in each of the 3 blocks. The consequence is two-fold: (*i*) execution time and energy jump to 28.7 ms and 2.0 mJ, respectively; (*ii*) the dimension of the modified TEMPONet grows to 970 kB, too large to be suitable for embedded deployment in the GAP8 L2 memory (512 kB).

For fair benchmarking, the approach must be compared against platforms capable of running deep learning algorithms (e.g., ARM Cortex-M or ARM Cortex-A family) on sEMG signals. There are some embedded systems for sEMG processing and gesture classification, such as [171] and [172], which can execute DL algorithms on a Cortex-A processor with a power envelope $\geq$ 500 mW, almost one order of magnitude larger than GAP8. Platforms of this class can run inference of DNNs [182], but their size and power envelope limit their applicability to embedded wearable systems. Recently, some attempts have also been made to deploy DNNs onto high-end ARM Cortex-M processors (e.g., ARM Cortex-M7 on STM32H7 MCUs), leveraging the energy-efficient software support provided by CMSIS-NN, the SoA library in the software implementation of DNNs. However, this kind of deployment reaches a top performance of 0.69 MAC/cycle @ 346 mW, 400 MHz measured on an STM32H7 MCU [170], more than 10$\times$ slower and 23$\times$ less efficient than the presented TCN implementation that combines parallel execution of the GAP8 cluster cores and the ISA extensions utilized by the PULP-NN computational backend [170].

4.3 Online Unsupervised Arm Posture Adaptation for sEMG-based Gesture Recognition on a Parallel Ultra-Low-Power Microcontroller

4.3.1 Overview

Despite the capabilities of automated learning, a major challenge to the long-term accuracy and robustness of sEMG-based control lies in the sEMG's inherent variability factors, such as anatomy, fatigue, skin perspiration, and electrode repositioning. This challenge was exposed in depth in Section 2.4.

SoA ML/DL approaches address this issue with two approaches: multi-session training or model adaptation. Multi-session training involves training a model on data acquired on diverse conditions, varying postures, electrode placement, or users [92]. Multi-session training is the method I extensively exploited in the previous contributions presented in this chapter to increase the generalization capabilities of non-deployed TCNs 4.1 and the fully deployed TCN TEMPONet 4.2 In contrast, model adaptation retrains part of a model on a new session's data. Both approaches are used in DL since (i) multi session-training was shown to benefit Deep Neural Networks (DNNs) more than non-deep ML [22], and (ii) DNNs' modularity allows to limit the retuning to few layers (such as Batch-Norms (BNs) in AdaBN [114], [116], or a net's backend in continual learning [183]).

Multi-session training and DNN retraining successfully improve generalization, but both come at high computation and memory costs. Multi-session training can only be done by saving multiple sEMG sessions and training a net on a server. On the other hand, adaptation by fine-tuning can partially mitigate the memory requirements thanks to latent replay techniques [184] (which only store a subset of inner activations [183]) or by only fine-tuning the final layers or BNs (avoiding back-propagation [114]); however, these methods still require computing complete net inferences.

Including an adaptation stage in the preprocessing steps can mitigate data variability by avoiding retraining the classification algorithms. For instance, Canonical Correlation Analysis (CCA) is used to linearly remap the sEMG signal of a new session to the values of the reference session [185]–[187]. The reference session is the set of data the classifier has initially seen in a full training, which can be done offline. However, a major shortcoming of this technique is the lack of online adaptation since (*i*) CCA is trained on sEMG steady gestures only, which requires high-accuracy segmentation of the new session even before adaptation; (*ii*) CCA training needs to store data segments of both the old session(s) and the new session; and (*iii*) CCA training is not implemented in real-time.

In this contribution, I target the sEMG arm posture variability by applying the adaptation to the preprocessing stage, classifying with a non-retrained classifier. In contrast to CCA-based approaches, I present a real-time adaptation method based on online Principal Component Analysis (PCA). I use online PCA to determine the principal components of the distribution of the new sEMG data and rotate the new data to match the new principal components to the ones of the old data. This approach overcomes all the limitations mentioned for the CCA works: (*i*) PCA is unsupervised and performed on all data, needing no training or segmentation based on the ground truth, which is not available in real learning in-the-wild scenarios [188]; (*ii*) the PCA is updated online on each sample, removing the need to store data and reducing the computational burden.

The contribution of this section is three-fold:

- I present an adaptation method using online PCA on sEMG, based on Oja's learning rule [189];
- I validate the strategy targeting the arm posture variability of the UniBo-INAIL dataset [92], getting a 37% to 51% recovery of the inter-posture accuracy drop;
- I deploy the method on the PULP MCU GAP9, showing a latency compatible with the sampling rate and channel count of SoA sEMG acquisition setups, even high-density, e.g., a latency of 0.843 ms for dimension 32, which keeps up with 32 channels at 1 kHz).

Overall, I combine unsupervised learning, online processing, and parallel computing to enhance embedded sEMG-based control systems in real-world scenarios.

I released the code implemented for this work.[1] As a research group, we also published the UniBo-INAIL dataset.[2]

[1] `https://github.com/pulp-bio/online-semg-posture-adaptation`
[2] `https://github.com/pulp-bio/unibo-inail-semg-dataset`

4.3.2 Materials & Methods

4.3.2.1 The UniBo-INAIL dataset

This contribution targets the UniBo-INAIL sEMG dataset, realized in a collaboration between the University of Bologna and the INAIL institute, first realized for the work [92]. A thorough introduction about sEMG-driven HMIs is provided in the background Section 2.4. Ethical approval was obtained from the local ethics committee, and all participants provided informed consent before participating in the study. The data are from 7 healthy male participants aged 29.5 ± 12.2 years, each undergoing 8 acquisition days with 4 sessions per day, each with a different arm posture: proximal (the sole with the arm not fully extended; the most common in literature), distal, distal with palm down, and distal with arm 45° up. Each of the 224 sessions is a complete dataset of 5 hand gestures typical of daily activities maintained for 3 seconds: power grip, two-finger and three-finger pinch grip, pointing index, and open hand; each repeated 9 to 16 times. Including rest positions (3 seconds), this protocol amounts to 6 classes. sEMG data were acquired at 500 samples/s via 4 Ottobock 13E200[3] sensors placed on the forearm muscles involved in the chosen gestures (*extensor carpi ulnaris*, *extensor communis digitorum*, *flexor carpi radialis*, and *flexor carpi ulnaris*).

4.3.2.2 Online PCA adaptation

PCA as adaptation The presented method uses PCA to identify the Principal Components (PCs) of the sEMG sessions' data distributions. We do not use PCA for dimensionality reduction, but the method can include it in general. The adaptation consists in (1) determining the PCs of the new session; then (2) using them in inference as a linear transformation that aligns the new session to the reference session:

$$\mathbf{x}' = \mathbf{W}_{\text{ref}}\mathbf{W}_{\text{new}}^{\mathsf{T}}\mathbf{x} \tag{4.5}$$

where $\mathbf{x} \in \mathcal{R}^{C\times 1}$ is the original data of the ongoing new inference session (1 sample per channel), with C the number of channels; $\mathbf{x}' \in \mathcal{R}^{C\times 1}$ is the transformed sample vector. $\mathbf{W}_{\text{ref}}, \mathbf{W}_{\text{new}} \in \mathcal{R}^{C\times C}$ are the PCA coefficient matrices of the reference session and the new session respectively, where PCs $\mathbf{w} \in \mathcal{R}^{C\times 1}$ are columns: $\mathbf{W} = [\mathbf{w}_1 \cdots \mathbf{w}_i \cdots \mathbf{w}_C]$. Multiplying by $\mathbf{W}_{\text{new}}^{\mathsf{T}}$ moves $\mathbf{x}$ to the basis where the covariance of the new session is diagonal; multiplying by $\left(\mathbf{W}_{\text{ref}}^{\mathsf{T}}\right)^{-1} = \mathbf{W}_{\text{ref}}$ projects the data to the reference session's space. This is analogous to the CCA-based alignment [187]; the advantage of PCA over CCA is that PCA matrices are orthonormal, so that inversion is just a transposition

[3] https://shop.ottobock.us/c/Electrode/p/13E200~550

requiring no decomposition. It is worth remarking that this adaptation strategy is unsupervised since it only deals with the distribution and covariances of the sEMG data $\mathbf{x}$, without requiring the ground truth class label, just like an actual calibration session has input data available but no knowledge about the gesture being executed. The following subsection deals with determining $\mathbf{W}_{\text{new}}$ online.

Oja's learning rule The Oja rule is an algorithm for determining the PCA coefficient on a stream of data by updating the coefficients upon reception of each sample $\mathbf{x} \in \mathcal{R}^{C \times 1}$ [189]. The steps for updating each component $\mathbf{w}_i$ are the following:

1. compute $y_i = \mathbf{w}_i^\intercal \mathbf{x}$;
2. compute the update $\Delta \mathbf{w}_i$ based on the learning rate λ:

$$\Delta \mathbf{w}_i = \lambda \cdot y_i \cdot \Big(\mathbf{x} - y_i \mathbf{w}_i - 2 \sum_{j<i} y_j \mathbf{w}_j \Big); \tag{4.6}$$

3. apply the update: $\tilde{\mathbf{w}}_i' = \mathbf{w}_i + \Delta \mathbf{w}_i$;
4. impose normalization: $\mathbf{w}_i' = \tilde{\mathbf{w}}_i' / \left\| \tilde{\mathbf{w}}_i' \right\|_2$.

This produces the new orthonormal coefficient matrix $\mathbf{W}' = [\mathbf{w}_1' \cdots \mathbf{w}_i' \cdots \mathbf{w}_C']$ updated on the fresh datum $\mathbf{x}$. Step 4 is required because the summation in Step 3 implements the orthogonality constraint but not the normalization of each vector. The two constraints hold after each update so that the Oja method requires no iterative matrix orthogonalization, inversion, or decomposition. Thus, the only iteration is over time on the stream of incoming samples.

Heuristics Applying the Oja rule successfully to sEMG sessions required the following heuristics:

- coefficients were initialized as $\mathbf{W}_{\text{ref}}$, i.e. the PCs of the reference session;
- the learning rate λ was scheduled as a function of the sample number t (natural dimensionless) as

$$\lambda_t = \frac{1}{1 + \beta t} \tag{4.7}$$

  with $\beta = 100$ (dimensionless);
- the new PCs $\mathbf{w}_{\text{new}}$ were reordered based on the best match with the reference components $\mathbf{w}_{\text{new}}$, and multiplied by -1 if the angle with the corresponding reference component was $> \pi/2$; It was observed that failing to apply this rematching caused the accuracy on new sessions to drop to chance level.

Parallelization For each $\Delta\mathbf{w}_i$, the orthogonalizer term in Equation 4.6 is a summation involving all previous $i-1$ components $\mathbf{w}_j$ for $j<i$. Hence, splitting over PCs, i.e., determining $\Delta\mathbf{W}_{\text{new}}$ by blocks of columns, creates an uneven load across workers. For this reason, the algorithm is parallelized along the PCs' coefficients, i.e., $\Delta\mathbf{W}_{\text{new}}$ is computed by blocks of rows, which is a workload evenly distributed across cores.

4.3.2.3 Classifier

The classification relies on the Multi-Layer Perceptron (MLP) with 8 hidden units used in the first paper on the UniBo-INAIL dataset [92]. The setup discards the Radial-Basis Function kernel Support Vector Machine, which proved 1% more accurate at the cost of a non-predictable model size after each training due to the varying number of support vectors. After training on a reference arm posture, the MLP is quantized (2.3.1) to 8-bit (i.e., `int8` weights and `uint8` activations) by applying PArameterized Clipping acTivation (PACT) [67] with the open-source library QuantLib [70]; then, the MLP is kept frozen in the experiments on unseen arm postures, where only the PCA undergoes adaptation. Refining the classifier is not the goal of this work; moreover, deeper convolutional models have yielded no accuracy benefit on the UniBo-INAIL dataset so far [190].

4.3.2.4 Experimental protocol

The analysis involves three kinds of experiments:

- Multi-posture training, as a high-accuracy baseline. For each subject, for each day, the classifier is trained on the first 5 repetitions of all gestures from all 4 arm postures, then validated on the following repetitions. This training shows the classifier all arm postures.
- Adaptation inter-posture, i.e., the proposed method. For each subject, for each day, each of the 4 arm postures is selected in turn as a reference, on which the classifier is trained using the first 5 repetitions of each gesture; then, in a nested fashion, each of the remaining 3 arm postures is chosen as arrival posture: PCA is adapted online on the first 5 repetitions of all gestures of the arrival arm posture, and validation is performed on the following repetitions.
- Bottom baseline: same data split as previous, without adaptation.

All gesture repetitions include the adjacent rest positions so that trainings, adaptations, and validations always contain all 6 classes, i.e., 5 gestures plus rest. All experiments

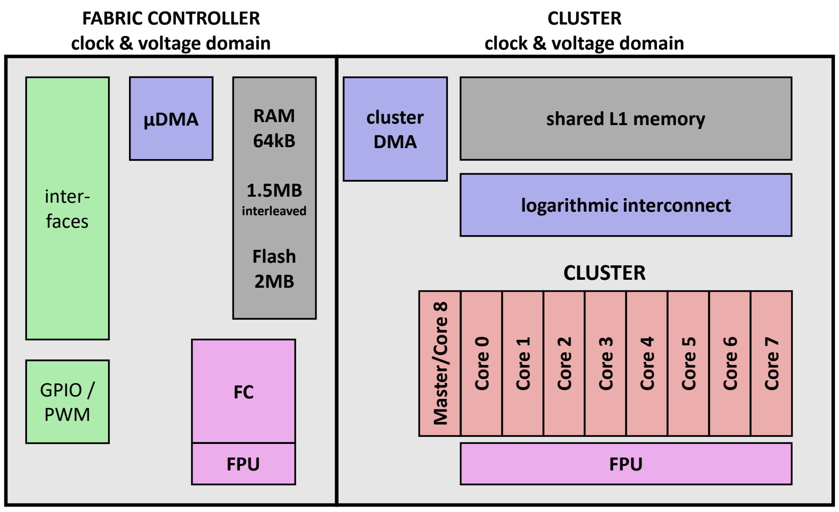

Figure 4.15: Scheme of the GAP9 MCU and its 8-core cluster.

are single-subject and single-day, with no multi-subject or multi-day training or inter-subject or inter-day validations.

4.3.2.5 Accuracy metrics

For each of the three experiments described in 4.3.2.4, the classification accuracy is measured as median ± Mean Absolute Deviation (MAD) over the experiments' repetitions, i.e., the nested loop selecting every subject, every day, every arm posture as a reference, and every remaining arm posture as the new target session. The MAD is defined as

$$\mathrm{MAD} = \operatorname*{median}_i \left(|a_i - \tilde{a}| \right) \tag{4.8}$$

where a_i's are each repetition's accuracy values and $\tilde{a}$ is their median. Median ± MAD is a more robust choice than mean ± standard deviation, motivated by the fact that the experiments deal with diverse data from different users and days.

4.3.2.6 Deployment & profiling on a parallel ULP MCU

The proposed online PCA method is deployed onto GAP9 (2.2) [34] (Figure 4.15), a commercial MCU mounting a PULP 9-core cluster accelerator based on the RISC-V Instruction Set Architecture extended with custom instructions [62], [63]. GAP9 is SoA in that it has the lowest energy consumption on MLPerf Tiny v1.0 benchmarks [39]. The algorithm is implemented in C, parallelizing on 8 cores as per 4.3.2.2. The MCU is configured to $V_{\mathrm{dd}} = 0.65\,\mathrm{V}$ and $f_{\mathrm{CLK}} = 240\,\mathrm{MHz}$, which is GAP9's best energy-efficiency configuration. The execution cycles were measured using the performance

Table 4.4: Results of the three protocols. Accuracy is reported as median ± MAD.

	DAY 1		**FULL DATASET**	
	Accuracy (%)	**Recovered fraction of the drop**	**Accuracy (%)**	**Recovered fraction of the drop**
multi-posture training	91.0 ± 2.9	-	91.2 ± 2.3	-
no adaptation	80.8 ± 4.8	-	81.8 ± 4.9	-
online Oja-based PCA adaptation	86.0 ± 4.2	0.510	85.3 ± 4.8	0.372

counter available in PMSIS[4], the open-source system layer for GAP9's operating system. Latency was determined as num_cycles/f_{CLK}. The power consumption was measured experimentally, and the energy consumption was determined as power$\times$latency.

4.3.3 Experimental Results

4.3.3.1 Classification accuracy

The results of the three experimental protocols described in 4.3.2.4 are reported in Table 4.4. As expected, multi-session training achieves the highest accuracy since all arm postures are seen at training time; however, it requires storing the data of all arm postures' sessions and performing a complete training, which is not feasible online on-device. The no-adaptation experiment yields the lowest accuracy and measures the inter-posture drop if no countermeasure is taken. As can be seen, the adaptation method relying on online Oja-based PCA recovers more than one-third (37%) of the accuracy drop on the whole dataset. Remarkably, the recovery is slightly above 50% on Day 1, the highest recovery across all dataset days. This difference might be related to user experience, which was minimum on Day 1; this effect will be investigated in future work.

4.3.3.2 Profiling

Figures 4.16 and 4.17 show the profiling results. Figure 4.16 shows the speed-up obtained running the Oja-rule update step and the normalization on 8 cores of the GAP9's cluster compared to execution on 1 core. The algorithm is profiled varying the number of channels to show how the method scales on a high sEMG channel count. The best speed-up obtained is 6.31×. For a low channel count, the speed-up is not

[4]https://greenwaves-technologies.com/manuals/BUILD/HOME/html/index.html

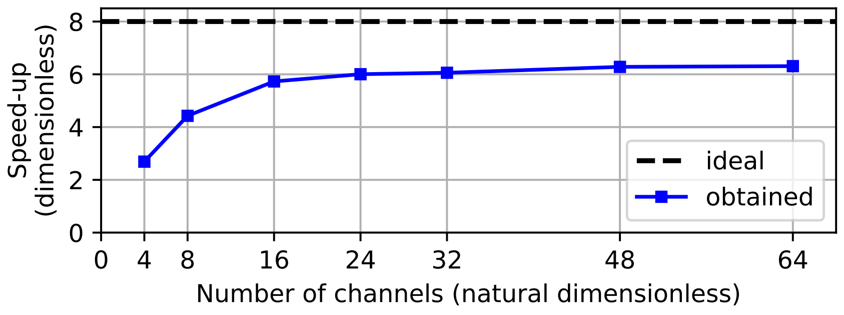

Figure 4.16: Profiling of the Oja-rule update step and normalization on the parallel ULP microcontroller GAP9: speed-up as a function of the number of sEMG channels.

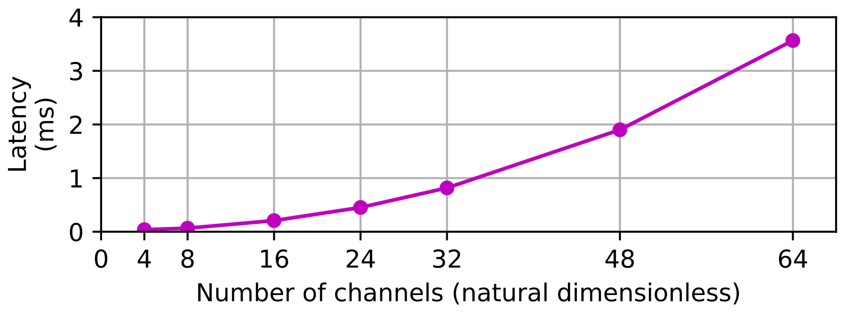

Figure 4.17: Profiling of the Oja-rule update step and normalization on the parallel ULP microcontroller GAP9: latency as a function of the number of sEMG channels.

optimal due to the small overall size of the computation, which is not an issue since few channels allow for very small latency. For the high channel counts reaching speed-up $> 6\times$, the non-ideality is due to the overhead of the memory transfers between the shared L2 memory and the cluster's private L1 memory, occurring before and after the computation. It is possible to observe that the obtained speed-up in the range $6.0 - 6.3\times$ is in the same range as the speed-up obtained by SoA PCA + Independent Component Analysis parallel implementations on the same number of cores, but for inference only [191].

Figure 4.17 shows the execution latency of the Oja-update and normalization. For channel counts up to 32, latency is 0.843 ms $<$ 1 ms, which means that the online solution can keep up with a sampling frequency of 1 kHz. For channel count > 32, the latency is longer, and the application must skip samples, causing a slower fit and making the calibration session longer. However, a $2\times$ or $3\times$ factor on the duration of the calibration session is a feasible compromise since calibration sessions can last a few minutes.

The measured power consumption is of 35.1 mW, which is 31% less than previous strategies based on retraining an embedded DNNs on-device [183]. As to energy consumption, an upper bound is represented by the 64-channel setup, which has a latency of 3.57 ms and thus consumes 0.125 mJ per update step. This energy consumption per update is 5.2× to 7.2× smaller than the typical energy per inference on previous embedded DNNs for sEMG [3], [183], proving the advantage of the presented solution over forward-pass techniques such as retraining of the last layers or AdaBN [114], [116].

Chapter 5

Regression: sEMG-based Estimation of Hand Kinematics and Force

This chapter presents the contributions of this thesis that belong to the task of multi-target regression. As Chapter 4, which dealt with multi-class classification, this chapter entirely belongs to the domain of sEMG-based hand modeling to advance non-invasive intuitive wearable HMIs driven by the sEMG signal processed by a low-power MCU alone. In this research field, regression is promising for enabling a more fluid and versatile control than recognition of fixed discrete hand positions, as exposed in 2.4.3. This chapter presents three contributions. The first work (5.1) presents an embedded TCN that accurately models hand kinematics by estimating the hand joint angles. The second work (5.2) addresses the estimation of hand joint angles by extracting event-based features. The third work (5.3) extends the previous heuristic to simultaneous multi-finger force estimation from the HD-sEMG. The event-based features exploited in the second and third contributions are computationally cheaper than TCNs and are promising for future porting onto event-driven devices with reduced latency and energy consumption.

5.1 sEMG-based Regression of Hand Kinematics with Temporal Convolutional Networks on a Low-Power Edge Microcontroller

5.1.1 Overview

In this section, I address the challenge of sEMG-based regression to decode hand kinematics. I present a regression framework based on a TCN (2.1), a DL model for time series modeling suitable for real-time operation on resource-constrained devices. I present the following contributions:

- I apply the SoA TEMPONet (4.2.2.2) TCN architecture on NinaPro DB8, a benchmark sEMG regression dataset also comprising trans-radial amputees, obtaining a Mean Absolute Error as low as 6.89°, which is 0.15° better than the dataset's SoA even if the model's bitwidth is reduced to 8 bit, while the SoA network is in `float32`.
- I further optimize the TCN, identifying the model size which yields the optimal tradeoff of regression error vs. memory footprint and MAC operations; regression accuracy is preserved, limiting the model size to 70.9 kB and the number of operations to 3.16 MMACs.
- I deploy the solution on the GAP8 edge microcontroller [34], measuring the performance in terms of latency and power consumption. I obtain 4.76 ms latency and 0.243 mJ energy cost per inference, demonstrating the suitability of the regression TEMPONet for edge low-power nodes working in real time.

5.1.2 Materials & Methods

5.1.2.1 NinaPro Database 8

The Non-Invasive Adaptive hand Prosthetics Database 8 (NinaPro DB8) [120], [192] is a public sEMG (2.4) database for finger position decoding, intended as a benchmark for estimation/reconstruction of kinematics instead of classification of gestures or grasps. In particular, contralateral movements are intended as a target for sEMG regression. NinaPro DB8 comprises 10 able-bodied subjects and 2 right trans-radial amputees. All participants repeat 9 kinds[1] of bilateral mirrored movements, lasting approximately

[1]Both single-finger and functional: thumb flexion/extension; thumb abduction/adduction; index finger flexion/extension; middle finger flexion/extension; combined ring and little fingers flexion/extension; index pointer; cylindrical grip; lateral grip; tripod grip.

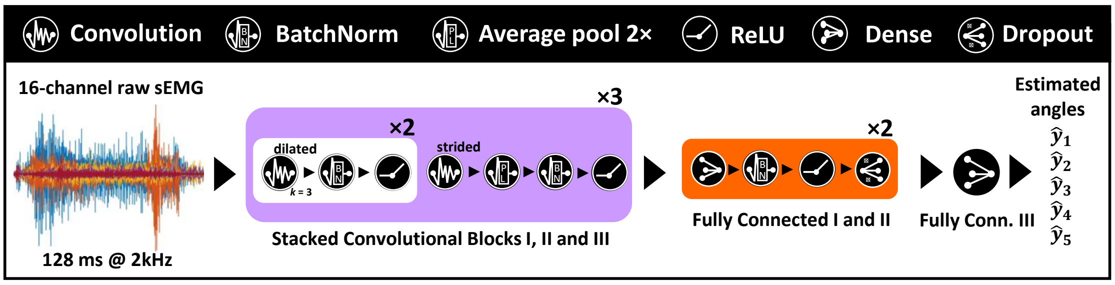

Figure 5.1: The TEMPONet TCN architecture, adapted for regression and further optimized compared to its first proposed version [3].

6 s − 9 s (i.e., slow on purpose, for transient modeling), interleaved with approximately 3 s of rest. The muscular potential was recorded using 16 active double-differential sensors (from a *Delsys Trigno IM Wireless EMG* system), positioned on two rows of eight units around the participants' right forearm in correspondence to the radiohumeral joint. Hand kinematics was acquired by an 18-DoF *Cyberglove 2* worn on the left hand, contralateral to the sEMG electrodes forearm, measuring the angles of the 18 dataglove joints. All signals were upsampled to 2 kHz and post-synchronized.

In this $\mathbf{X}(t) \mapsto \mathbf{Y}(t)$ multivariate regression formulation, the input information is the 16-channel sEMG signal, and the target is represented by 5 DoA, defined as linear combinations of the 18 DoF of the glove [120]. This DoF-to-DoA reduction serves to (*i*) discard or downscale irrelevant DoF, and (*ii*) directly target DoA defined as relevant hand movements. The SoA on NinaPro DB8 is represented by the LSTM of [121], which attains an MAE of 7.04°. (The work [193] is more recent, but it is not SoA since it does not address the 2 thumb DoA of the dataset, but only tackles the remaining 3, namely, index, middle, and ring+little, without providing any justification for this limitation.) However, the LSTM proposed in [121] is not suitable for deployment on low-power embedded platforms due to its `float32` numeric format. Furthermore, the authors do not report the essential information about the number of hidden units, parameters, and MAC of the LSTM employed. Moreover, LSTMs are, in general, more difficult to train than convolutional models [15], a further downside that motivates the exploration of Temporal Convolutional Networks, as explained in 2.1.

5.1.2.2 TEMPONet architecture for regression

The network presented in this contribution is a further development of the SoA TEMPONet TCN (4.2.2.2) [3]. In addition to modifying it to target regression, a model exploration is presented to identify the best tradeoff between accuracy and network size. The net's architecture, shown in Figure 5.1, features 3 Convolutional Blocks, each stacking:

- 2 dilated causal convolutions with kernel size 3, variable dilation d, and full padding;
- 1 convolution with kernel size 5, variable stride s, followed by an average pooling (kernel 2, stride 2).

The 3 convolutional blocks have dilation $d = 2, 4, 8$ and stride $s = 1, 2, 4$, respectively. This novel exploration for a more efficient TEMPONet reduces the channels of the 3 convolutional blocks compared to the original model of (4.2.2.2) [3]. This exploration halves Block I's channels from 32 to 16, halves Block II's channels from 64 to 32, and explores different channel numbers for Block III, namely, 32, 48, 64, 96, 128, and 192, searching for the best tradeoff between regression error and deployment metrics, i.e., model size, MAC, latency, and energy consumption. This model search on TEMPONet is novel since it is not performed in the paper first proposing it (4.2.2.2) [3]. After the convolutional blocks, 3 FC layers perform the classification. FC I has $4\times$ units as Block III's channels (variable number as explained above), FC II has 32 units, and FC III has 5 units, corresponding to the 5 DoA target of the regression. All layers, except FC III, have ReLU non-linearity as the activation function and are equipped with BN to counter internal covariate shift [117]. FC I and FC II are trained with dropout with $p_{\text{drop}} = 0.5$, to help regularization [165]. The presented optimized TEMPONet processes a 256 samples input window (128 ms @ 2 kHz) with less than 500 k parameters. It is fed by raw signals, thus with no preprocessing or feature extraction overhead. Size and computation improvements of the final selected architecture compared to the original TEMPONet are detailed in Subsection 5.1.3.

5.1.2.3 Experimental setup details

Dataset split For each of the 12 subjects of NinaPro DB8, 3 sessions are provided. As recommended by the dataset's authors, session 3 (2 repetitions per movement) was used as test set. Sessions 1 and 2 (10 repetitions per movement) were merged and used in a 2-fold cross-validation setup.

Preprocessing The NinaPro DB8 signals made available at [192] have already been bandpass-filtered with a 4th-order Butterworth between 10 Hz and 500 Hz. Classical ML models, namely SVM and MLP, which need feature extraction on every signal window, were trained on the WL feature extracted from 60 ms-windows of each channel, with a slide of 100 ms for SVM (largest computationally affordable training set size) and 25 ms for MLP. For the SVM, the RBF kernel was used, applied to the data scaled to unit variance, and the C coefficient was tuned separately for each target DoA. TCNs were directly trained on raw sEMG signals, using time windows of 128 ms (i.e., 256 samples @ 2 kHz).

Machine learning setup Models were implemented in Python 3.8, using Scikit-learn 0.23 [166] for SVM and MLP and PyTorch 1.6 [65], [66] for TCN. The SVM is an RBF kernel SVM, and the MLP was implemented with 3 hidden layers. TCNs were trained with MAE loss, AdaM optimizer, initial learning rate $1 \cdot 10^{-4}$, and minibatch size 64. First, 19 epochs were run in `float32` format; then, post-training quantization to 8 bit was performed, and 1 last epoch of quantization-aware training was run, using PArameterized Clipping acTivation (PACT) [67] as implemented by NeMO (2.3.2) [43], [64], an open-source library for CNN minimization to target deployment on highly memory-constrained ultra-low power devices.

Model output postprocessing The outputs of all models were post-processed with an Exponential Moving Average (EMA):

$$y'_t = \alpha_{\text{EMA}} \cdot y'_{t-1} + (1 - \alpha_{\text{EMA}}) \cdot y_t, \tag{5.1}$$

with y and y' the unfiltered and filtered signal, respectively, t the time index, and $\alpha_{\text{EMA}} \in [0, 1]$ the decay factor. The decay factor α_{EMA} was tuned per-subject, per-DoA, for each model (after the model's training), using only training data and optimizing by grid search. Although the formula is differentiable, optimization by PyTorch's automatic differentiation plus SGD proved slower and less accurate than grid-search sweep. This optimization is consistent with the ML setup (since no test data are used) and is performed to tune the compromise between the EMA's beneficial smoothing and the inertia deriving from the weight of past values. It is worth remarking that the EMA is computed using only outputs y_t from the present and the past; thus, it adds no delay to the setup.

5.1.3 Experimental Results

5.1.3.1 Evaluation metrics

Assessing the effectiveness of a regression framework on the end-to-end control of a robotic hand is not a trivial task. The evaluation of the Mean Absolute Error (MAE) is important, but it is necessary to consider, for instance, that a difference between the value of estimated angles and the ground truth has a greater impact on control when it occurs during movements rather than in static phases. For this reason, the models' regression quality is measured by MAE (measured in degrees) and by a *regression accuracy* defined as the frequency of the MAE being below a tolerance Θ_{tol}:

$$\Theta_{\text{tol}}\text{-accuracy} \triangleq \frac{1}{T}\sum_{t=1}^{T} \mathbb{I}_{\Theta_{\text{tol}}}(\text{MAE}(t)) \tag{5.2}$$

where $\mathbb{I}$ denotes the indicator function:

$$\mathbb{I}_{\Theta_{\text{tol}}}(\text{MAE}(t)) \triangleq \begin{cases} 1 & \text{if } \text{MAE}(t) < \Theta_{\text{tol}} \\ 0 & \text{otherwise.} \end{cases} \tag{5.3}$$

A regression-accuracy threshold $\Theta_{\text{tol}} = 10°, 15°$ is empirically selected. These metrics are reliable measures of the end-to-end quality of the control for several reasons: (*i*) they are related to the actual scale of angular positions; (*ii*) they are statistically representative since averages are taken over joints, movements, and subjects, including two trans-radial amputees; (*iii*) MAE is first-order, hence less affected by outliers than the regression R^2. The regression accuracy, based on a threshold, is even more robust.

5.1.3.2 Models comparison

The results shown in Table 5.1 and Figure 5.2 report the tested ML (SVM and MLP) and DL (TEMPONet) algorithms along with the LSTM of [121], used as a baseline. In particular, Figure 5.2 depicts the search of the optimal size for the regression TEMPONet, varying the channels of Block III over the values 32, 48, 64, 96, 128, 196 (as explained in 5.1.2.2).

Table 5.1, shows that conventional ML frameworks can not match the state-of-the-art accuracy: the SVM reaches 7.28° (i.e., +0.24° compared to SoA), while the MLP obtains an MAE of 7.14° (i.e., +0.10° compared to SoA). Moreover, the SVM has two further limitations: (*i*) trying to improve the SVM's accuracy by increasing the training set size proved unfeasible due to diverging training time; (*ii*) even with the tuned C's, which regulate the bias-variance tradeoff per-DoA, the SVM incorporates on average 98.3% of the training examples as support vectors (16-dimensional), which amounts to 1.15 MB of memory, which is demanding for embedded devices with strict memory constraints. Note that the latter problem is an inherent methodological limitation of SVM, whose size can not be fixed a priori before training.

TCNs are the only model which proved capable of outperforming the SoA MAE. All reported TEMPONet's results refer to networks quantized to 8 bit format, which reduces the memory footprint by 4× compared to `fp32`. As can be seen from Figure 5.2, the smallest and largest TEMPONet show higher errors, indicating that they produce underfitting and overfitting, respectively, thus identifying the best bias-variance tradeoff in the in-between interval $\{64, 96, 128\}$. When Convolutional Block III has 64, 96, or 128 channels, TEMPONet's regression is equally accurate. Remarkably, the baseline is surpassed even when operating at a lower precision. Table 5.1, reports the MAE and regression-accuracy of the TEMPONet-64 against the LSTM of [121] and the SVM and MLP implemented in this contribution. In particular, with 64-channel Block III, the TEMPONet has an elbow in the curves regarding model size and MAC, thus representing the best MAE-vs-deployment tradeoff. This 64-channel TEMPONet has a memory footprint of just 70.9 kB and requires the computation of 3.16 MMACs, which represent a memory reduction of 6.5× and a computation reduction of 5.3× compared to the original TEMPONet proposed in (4.2) [3].

Figure 5.3, showcases an example of the regression output provided by this network. In particular, it is possible to observe that the output is prompt and accurate for both narrow and wide movements. Typical errors fall into two main categories. The first kind of error is an offset, stationary during each movement; since the output is stationary as well, this constant difference is not expected to affect the user's perceived accuracy; if perceived, offsets can be easily compensated via session-specific recalibration. The second kind of error is represented by fast erratic segments, which could be smoothed out by strengthening the EMA postprocessing; the optimal amount of EMA smoothing was optimized as explained in 5.1.2.3, to tune the smoothing-delay tradeoff best for the MAE; the average decay factor obtained was $\alpha_{\mathrm{EMA}} = 0.862$, with a standard deviation of 0.044 across subjects and DoA.

Finally, the experiments implemented the 64-channels TEMPONet on the commercial microcontroller GAP8 [34] to measure latency and energy cost per inference. For deployment, the setup used the open-source tool DORY (Deployment ORiented to memorY (2.3.2) [41], [42], [78]), with an extension to the backend to support dilated convolutions. Computation latency and energy consumption were measured experimentally, running inferences on GAP8. The energy consumption E_{exp} was determined by experimentally measuring the current consumed, $i_{\mathrm{exp}}(t)$, then integrating over the computation time interval:

$$E_{\mathrm{exp}} = V_{\mathrm{DD}} \int_0^{T_{\mathrm{exp}}} i_{\mathrm{exp}}(t)\mathrm{d}t, \tag{5.4}$$

where V_{DD} is the supply voltage and $[0, T_{\mathrm{exp}}]$ is the latency time interval required for the inference, measured experimentally. When running at $V_{\mathrm{DD}} = 1\,\mathrm{V}$, $f_{\mathrm{CLK}} = 100\,\mathrm{MHz}$ (the most energy-efficient configuration), GAP8 has a power consumption of 51.0 mW. This configuration yields a latency of just 4.76 ms per inference, with an energy cost of just 0.243 mJ per inference. These values demonstrate that the selected 64-channel TEMPONet can fit the strict constraints of resource-limited controllers and real-time

Table 5.1: Regression quality of the explored models, compared to the SoA of the NinaPro DB8.

Model	Format	MAE	10°-accuracy	15°-accuracy
SVM	`fp32`	7.28°	0.795	0.883
MLP	`fp32`	7.14°	0.799	0.889
LSTM [121]	`fp32`	7.04°	n.a.[1]	n.a.[1]
TEMPONet[2]	`int8`	**6.89°**	**0.814**	**0.900**

[1]The Correctness Score (CS) of [121] is accuracy, but is somewhat cherry-picked since per-joint tolerances are fixed as percentiles after dynamic range clipping.
[2]Best one selected: Block III with 64 channels.

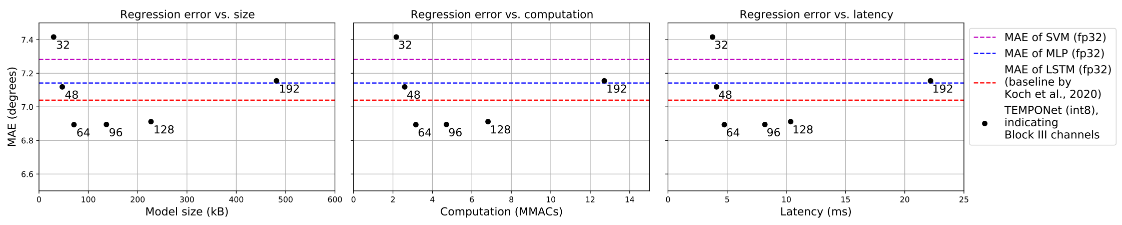

Figure 5.2: Deployment metrics of the novel optimized TEMPONet on GAP8 [34], as a function of the number of channels of Convolutional Block III.

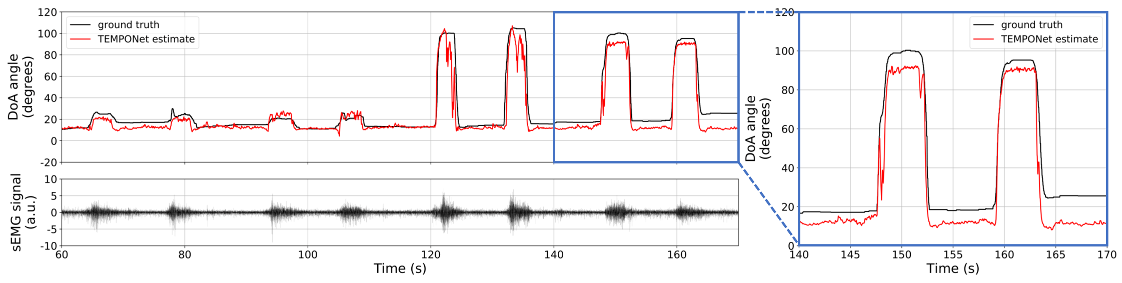

Figure 5.3: Example of the regression produced by the novel optimized TEMPONet: test session of Subject 1, DoA angle 5; sEMG signal from sensor 1 shown for reference.

operation. Regarding latency, the consensus on real-time requirements for artificial hand control is 300 ms [103]. Accounting for the 128 ms input window length, plus the 4.76 ms computation latency, the application matches real-time requirements with a wide margin, proving capable of providing a fluid control without a relevant perceived delay.

5.2 Event-based Low-Power and Low-Latency Regression Method for Hand Kinematics from Surface EMG

5.2.1 Overview

Finding an effective mapping from sEMG to control commands (2.4) is not a trivial task, and SoA approaches tackle the problem by resorting to ML or DL [106] where sEMG signals are mapped to a given set of gestures (classification) [22] or to continuous degrees of freedom (regression) [24]. As opposed to conventional pattern recognition, deep neural networks can learn signal features at training time, often outperforming the handcrafted features needed for non-deep ML; the learned information extraction is potentially optimal since data-driven, but not easily explainable.

Current research aims for a more natural and intuitive control, hence the need to move from a limited set of predefined positions to a continuous control that can be performed with a regression-based approach (2.4.3). So far, regression approaches to sEMG represent a minority, whereas the vast majority of the literature is concerned with classification, focusing mostly on DNNs to counteract the inter-session variability of sEMG and make classification robust in the long run through regularization [22] or adaptation [194]. In contrast, regression works are still scarce in the literature compared to robust classification efforts. Most existing proposed solutions for sEMG regression produce DL models that yield a low regression error [195], [196] but do not take fully into account the memory, latency, and energy constraints of resource-constrained embedded computational devices. Some proposed deep networks require processors designed explicitly for linear algebra [24] to be executed with ultra-low power consumption (tens of milliwatts power envelope), a regime suitable for long-term wearable devices.

In the veins of exploring solutions for ultra-low-power execution of computationally demanding tasks, Spiking Neural Networks (SNN) are an emerging class of artificial neural networks specifically designed to process data in the format of spike trains (i.e., binary with sparse 1's, coming as a stream in continuous physical time). Neurons in an SNN have a state that emulates the biological membrane potential: it is excited at the reception of events and decays over time; upon crossing a threshold, a neuron *fires*, i.e., transmits an event (a *spike*) to the connected neurons [197]. Neuromorphic processors, either digital [198], [199] or mixed-signal [200], are an ideal substrate for deploying SNNs since they are accelerators for the sequential computation of the emulated potential varying in time. Moreover, neuromorphic processors perform event-proportional computing [198]: since each neuron influences the others only when it fires, events are sparse spike trains, which cause only sparse updates of the net's neurons' states, greatly reducing latency and energy consumption compared to inference in a conventional neural network that requires to compute the entirety of activation maps.

Integration of sEMG acquisition systems with neuromorphic processors requires converting the digital sEMG raw data into sequences of events, i.e., timestamps associated with each channel to be used as input spike train for the event-based processing. Existing works on processing sEMG on event-driven hardware show that separation patterns can be extracted with SNNs consuming $< 1\,\text{nJ}$ per spike, amounting to as little as $0.05\,\text{mW}$ (hundredths of a milliwatt) of total power [201]. However, these works do not address regression yet, but implement classification [201]–[203] or provide insight into the empirical activation patterns and their class-separability [204].

In this section, I present a hybrid method based on event-based EMG encoding, a bio-inspired feature extraction, combined with regression implemented on a low-power processor ideal for embedded solutions. The purpose is to explore a goodness-complexity tradeoff for sEMG regression against the existing literature that focuses on DL models [24], [193], [195], [196]. My contribution is three-fold:

- I present an event-based encoding strategy for the sEMG that works in streaming, i.e., consumes inputs one by one, producing a stream of output spike events;
- I tune the encoding scheme on the real sEMG regression dataset NinaPro DB8 (5.1.2.1), achieving a Mean Absolute Error of 8.8 ± 2.3 degrees, comparable with the SoA DNN, proving that the spike conversion preserves enough information for a fine task like regression;
- I profile the resource requirements and execution of the setup on a commercial digital microcontroller, getting $9\times$ smaller memory footprint, $10\times$ shorter latency, and $13\times$ lower energy consumption per inference compared to the SoA deep net, proving the method a perfect fit for resource-constrained embedded platforms.

I release open-source the code developed for this research.[2]

5.2.2 Materials & Methods

5.2.2.1 Encoding surface EMG to events

This contribution performs EMG-to-spike conversion, i.e., encoding of the sEMG data to an event-based format, which is a simplification of the cochlear method. Cochlear

[2] https://github.com/pulp-bio/event-based-semg-regression

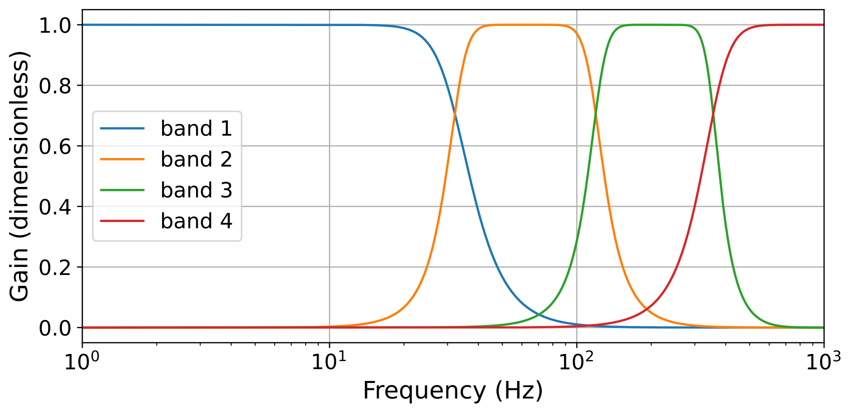

Figure 5.4: Gains of the four 4-th order Butterworth filters of the used frequency bands. They stop at 1 kHz since it is the Nyquist frequency of the NinaPro DB8 dataset.

spike conversion is a signal processing pipeline inspired by the mechanical separation of acoustic frequencies taking place in the mammalian cochlea prior to transduction and encoding into neural train spikes. The natural cochlea has stimulated bio-inspired hardware designers to implement silicon cochleas, i.e., circuitry mimicking the natural spike conversion, either in digital [205] or analog solutions [206], [207]. The method consists of two stages: (1) bandpass filtering and (2) Leaky Integrate-and-Fire (LIF) neurons.

1. Bandpass filtering Each input channel is passed into a bank of bandpass filters. The filtering used 4 bands covering the whole bandwidth from 0 to f_{Nyquist}, based on 4-th order Butterworth filters. The cutoff frequencies are set with exponential spacing (adjusted to reach zero):

$$f_n = \frac{e^n - 1}{e^{N_{\text{bands}}} - 1} \cdot f_{\text{Nyquist}} \qquad n = 0, \cdots, N_{\text{bands}} \tag{5.5}$$

where $N_{\text{bands}} = 4$ is the number of bands and f_n is the n-th cutoff frequency between adjacent bands. Since NinaPro DB8 has $f_{\text{sample}} = 2\,\text{kHz}$ and thus $f_{\text{Nyquist}} = 1\,\text{kHz}$, the resulting cutoff frequencies are

$$f_{0,1,2,3,4}\,[\text{Hz}] \;=\; 0.0,\; 32.1,\; 119.2,\; 356.1,\; 1000.0 \tag{5.6}$$

The gains of the 4 4-th order Butterworth filters corresponding to the 4 bands are shown in Figure 5.4. The NinaPro DB8 dataset is released already filtered with a 4-th order bandpass Butterworth between 10 Hz and 500 Hz, and no additional cleaning was applied prior to splitting bands. Passing each input channel in the 4 filters expands the number of input channels from 16 to 64. Then, all signals are full-wave-rectified.

2. LIF neurons The spike conversion in the strict sense takes place in 64 LIF neurons, each receiving the output of one filter as an injected input current. LIF neurons are a simplified model of the natural neuron, where the state is described by a *membrane potential* $V_{\mathrm{mem}}(t)$ obeying the linear electrical law

$$\frac{\mathrm{d}V_{\mathrm{mem}}}{\mathrm{d}t} = -\frac{(V_{\mathrm{mem}}(t) - \mathcal{E}_{\mathrm{leak}}) - \frac{I_{\mathrm{inj}}(t)}{g_{\mathrm{leak}}}}{\tau} \tag{5.7}$$

where τ is the membrane relaxation time, $\mathcal{E}_{\mathrm{leak}}$ is the leak reversal potential, I_{inj} is the injected current, and g_{leak} is the leak conductance. Each time V_{mem} crosses a threshold V_{thr}, the neuron emits a spike event corresponding to the time of crossing t_{fire}. After each spike the LIF neuron undergoes a *refractory time* t_{refr}, i.e. a time interval $[t_{\mathrm{fire}}, t_{\mathrm{fire}} + t_{\mathrm{refr}}]$ during which V_{mem} is forced to a reset value V_{reset}, and neither the decay nor the inhomogeneous driving term $I_{\mathrm{inj}}(t)/g_{\mathrm{leak}}$ act:

$$V_{\mathrm{mem}}(t) \equiv V_{\mathrm{reset}} \qquad t \in [t_{\mathrm{fire}}, t_{\mathrm{fire}} + t_{\mathrm{refr}}]\,. \tag{5.8}$$

The clearer change of variables

$$x(t) \triangleq \frac{V_{\mathrm{mem}}(t) - \mathcal{E}_{\mathrm{leak}}}{V_{\mathrm{thr}} - \mathcal{E}_{\mathrm{leak}}} \tag{5.9}$$

$$x_{\mathrm{drive}}(t) \triangleq \frac{1}{V_{\mathrm{thr}} - \mathcal{E}_{\mathrm{leak}}} \cdot \frac{I_{\mathrm{inj}}(t)}{g_{\mathrm{leak}}} \tag{5.10}$$

cleans away all electrical quantities (unneeded in a numerical simulation) and yields a dimensionless state with firing threshold $x_{\mathrm{thr}} = 1$ and law

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -\frac{x(t) - x_{\mathrm{drive}}(t)}{\tau} \tag{5.11}$$

making it clearer that the injected current plays the role of an external driving term. Each of the 64 filtered signals is used to drive an independent LIF neuron, which carries out the spike conversion. The LIF neurons are parameterized based on three values: (*i*) the empirical gain g_{data} to convert the arbitrary-units filtered signals into $x_{\mathrm{drive}}(t)$:

$$x_{\mathrm{drive}}(t) = g_{\mathrm{data}} \cdot |x_{\mathrm{bandpassed}}(t)|\,; \tag{5.12}$$

The grid-search explored 5 values from $1.0 \cdot 10^5$ to $1.0 \cdot 10^6$, approximately exponentially spaced:

$$g_{\mathrm{data}} \in \left\{1.0 \cdot 10^5, 1.7 \cdot 10^5, 3.0 \cdot 10^5, 5.5 \cdot 10^5, 1.0 \cdot 10^6\right\} \tag{5.13}$$

(*ii*) the membrane relaxation time, which was set to $\tau = 10\,\mathrm{ms}$; (*iii*) the refractory time, for which the grid-search explored $t_{\mathrm{refr}} \in \{1\,\mathrm{ms}, 2\,\mathrm{ms}\}$.

Table 5.2: Settings explored to tune the event-based encoding.

Parameter role	symbol	Explored values
gain from data to dimensionless LIF driving term x_{drive}	g_{data}	$1.0 \cdot 10^5$, $1.7 \cdot 10^5$, $3.0 \cdot 10^5$, $5.5 \cdot 10^5$, $1.0 \cdot 10^6$ (dimensionless)
refractory time of LIF neurons	t_{refr}	1 ms, 2 ms
decay time of post-LIF causal exponential kernel	τ_{post}	1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000 milliseconds

To feed spike trains to regression, the setup adopted a rate encoding. The firing rates were computed based on a causal exponential decay kernel [208] assigning each spike a weight of $\exp(-t/\tau_{\text{post}})$ for $t \geq 0$ in the future. In contrast with discrete windowing, this kernel does not require storing a variable buffer of events timestamps but only one floating-point number. This causal exponential-kernel rate can take values ≥ 0 and $< 1/\left(1 - \exp\left(-t_{\text{refr}}/\tau_{\text{post}}\right)\right)$. The grid-search explored 12 values from 1 ms to 5 s, approximately exponentially spaced:

$$\begin{aligned}\tau_{\text{post}}\,[\text{ms}] \in \{&1, 2, 5, 10, 20, 50, 100, 200, 500,\\ &1000, 2000, 5000\}.\end{aligned} \tag{5.14}$$

For clarity and compactness, all the explored settings of the event-based encoding are reported in Table 5.2.

LIF dynamics were implemented in Python 3.8 with the simulator Brian2 [209] v. 2.5 and run with simulation timestep 500 µs, found to be a good tradeoff between time resolution and computation time; since this is equal to the sampling time of the data, each filtered sample drives its channel's LIF for exactly 1 simulation step.

5.2.2.2 Regression

The experiments use linear regression to prioritize low memory and computation requirements over refined accuracy. The setup involved no preprocessing of the ground-truth DoA signals nor postprocessing of the regression outputs. The $\mathbb{R}^{64} \to \mathbb{R}^5$ multivariate regression problem is parameterized by a 5×64 coefficients matrix plus a 5-valued intercept; in `fp32`, this amounts to a memory footprint of 1576 bytes (including input and output), and a number of operations of 645 FLOP. As suggested by NinaPro DB8's authors, sessions 1 and 2 (10 repetitions of each gesture) were merged and used for training and validation, and session 3 (2 repetitions of each gesture) was used for testing. All experiments were performed separately for each subject, without any multi-subject training or inter-subject validation. For training, the spike trains were downsampled with a time step of 100 ms to keep the training computation time $<$ 8 hours for the whole dataset; at testing, the inference was called on the spike trains every 16 ms, which is the same time step as the SoA work [24] used as a baseline.

5.2.2.3 Profiling

The proposed processing was profiled based on memory footprint, number of operations, power consumption, latency per inference, and energy consumption per inference. To measure the latency per inference and to determine the energy consumption per inference, the setup implemented the spike conversion (filter banks + LIF neurons) and the linear inference on an STM32 F401RE microcontroller, which mounts an ARM Cortex-M4 processor. STM32 F401RE is not designed for event-based sparse data processing and does not implement event-driven execution. The motivation for profiling on this platform is to showcase how memory-, time-, and energy-efficient the proposed setup is, even when run on a commercial general-purpose microcontroller.

It was programmed in C and compiled with optimization `-Ofast`. Latency was measured using the debugger of the environment STM32CubeIDE v. 1.11, whose overhead produces a variability of cycle counts in the order of 10 cycles; at a clock frequency of 84 MHz, this amounts to an uncertainty of 0.1 µs, which is accurate enough for the purpose. Power consumption was determined based on the value of 146 µA/MHz reported in the datasheet[3]; at clock frequency 84 MHz with a power supply of 3.3 V, this amounts to a power consumption of 40.5 mW. Energy per inference was determined by multiplying by experimentally measured latency.

Subsection 5.2.3 shows that the pipeline can run in streaming, i.e., consuming one sEMG sample per channel at a time, fitting 10 update steps for each of the 64 LIF neurons within a latency of 500 µs, which is the sampling rate of the NinaPro DB8 dataset. Porting to actual event-driven neuromorphic devices (either digital [198] or mixed-signal [200]) will constitute a future work.

[3]`https://www.st.com/en/microcontrollers-microprocessors/stm32f401re`

5.2.3 Experimental Results

5.2.3.1 Evaluation metrics

The regression is evaluated using the Mean Absolute Error (MAE), measured in degrees and defined as

$$\mathrm{MAE} = \frac{1}{N_{\mathrm{infer}} N_{\mathrm{DoA}}} \sum_{i=1}^{N_{\mathrm{infer}}} \|\hat{\mathbf{y}}_i - \mathbf{y}_i\|_1 \tag{5.15}$$

where $\mathbf{y}_i, \hat{\mathbf{y}}_i \in \mathbb{R}^5$ are the multivariate ground truth and estimation, measured in degrees, corresponding to the i-th inference, respectively, $\|\cdot\|_1$ is the L_1-norm, $N_{\mathrm{DoA}} = 5$ is the number of DoAs, and N_{infer} is the total number of inferences in each subject's Session 3, obtained by calling one inference every 16 ms (as explained in 5.2.2.2). The MAE is a reliable metric of the end-to-end control goodness because it has the same scale as the target joint angles; moreover, MAE is a first-order statistic, thus more robust against outliers compared to (R)MSE or the multivariate coefficient of determination R^2, which are quadratic. All MAEs are averaged over time (hence over movement types and repetitions) and over DoAs, as expressed by Equation 5.15; and over all the 12 subjects; this allows comparison with [24]. The processing pipeline is profiled by determining memory footprint, number of operations, latency per inference (in cycles and milliseconds), and energy per inference, as explained in 5.2.2.3.

5.2.3.2 Regression accuracy

The regression results are reported in Table 5.3, which displays the optimal decay time $\tau_{\mathrm{post}}^{\mathrm{best}}$ for the causal exponential kernel $\exp(-t/\tau_{\mathrm{post}})$ and the corresponding MAE, obtained for the explored values of the data gain g_{data} and the LIF refractory time t_{refr}, as detailed in 5.2.2.1. These results provide insights both into the best settings and general trends.

Focusing on the optimal settings $g_{\mathrm{data}}^{\mathrm{best}} = 3.0 \cdot 10^5$ and t_{refr} equal to 1 ms or 2 ms, Figure 5.5 shows the curve for the search on τ_{post}, which identified the optimal value of the causal exponential kernel length, i.e., $\tau_{\mathrm{post}}^{\mathrm{best}} = 500\,\mathrm{ms}$. The curve is convex because a too-short (respectively, too-long) decay time of the exponential kernels weighs too much the recent (resp., old) spikes, i.e. computes the rate at a timescale that does not match the actual timescale of the hand's kinematics. The best MAE of 8.84 ± 2.26 degrees is consistent within 1 standard deviation with the SoA value of the TCN in [24], which is 6.89 ± 2.08 degrees. The TCN's MAE standard deviation is not reported in the SoA paper [24] and was determined by reproducing the setup of [24]. Compatibility within

1 standard deviation suggests that the setup yields a regression quality as good as the SoA from a practical point of view. The MAE standard deviations in the range from 2.0 degrees to 2.6 degrees are an empirical measure of the general, natural variability in angular regression error across DoAs and subjects; in contrast, the literature reporting standard deviations of regression accuracy reports only the range [120] or the standard deviation [193] of the determination coefficient R^2, which is dimensionless and standardized by DoA, hence not informative about the physical scale of the error variability.

Regarding the general trends outside the best settings, it can be seen that the choice of the refractory time has little impact compared to the choice of data gain since the former always produces differences in MAE of less than 0.1 standard deviations, and the range of MAEs obtained is mainly due to the data gain setting. Another strong general trend is the consensus about $\tau_{\text{post}}^{\text{best}} = 500\,\text{ms}$ since even the grid-adjacent values of 200 ms or 1000 ms never turn out to be optimal; this consensus is interpreted as a general estimation of the characteristic time scale of the variation of the kinematics in the NinaPro DB8 data.

It is worth remarking that τ_{post} by no means involves a delay in computation due to a wait. The decay due to the causal exponential kernel is applied recursively at every update of the firing rate: at each simulation step, the rate is updated by multiplying it by $\exp\left(-\Delta t_{\text{sim}}/\tau_{\text{post}}\right)$, where Δt_{sim} is the simulation timestep, then incrementing by $+1$ if the corresponding LIF neuron has fired at the current simulation step. This sequential update is performed at each step, requiring a lightweight computation, regardless of the numerical value of τ_{post}. In detail, since $\exp\left(-\Delta t_{\text{sim}}/\tau_{\text{post}}\right)$ is constant and is precomputed once for all, updating the rate r as $r \leftarrow r \cdot \exp\left(-\Delta t_{\text{sim}}/\tau_{\text{post}}\right) + 1\,[\text{if spike}]$ only requires 1 multiplication, 1 condition test, and (if test if positive) 1 addition per neuron per simulation step. A long τ_{post}, such as $\tau_{\text{post}}^{\text{best}} = 500\,\text{ms}$, simply means that the relative decrease per simulation timestep is small.

5.2.3.3 Profiling

The profiling results on the STM32 F401RE commercial microcontroller are reported in Table 5.4. Compared with the TCN of [24], the proposed regression setup has 9.8× smaller memory footprint, 10.6× shorter latency, and 13.3× lower energy consumption per inference. Thus, the event-based processing proposed in this work achieves an accuracy comparable to the SoA [24] while fitting in a much more limited resource budget. The memory saving is due to the fact that the application does not buffer temporal data, always consuming data in streaming except for the filters' states of the initial filtering stage. In particular, it is noteworthy that the latency of $\lesssim 450\,\mu\text{s}$ per inference

Table 5.3: Best decay time $\tau_{\text{post}}^{\text{best}}$ of the causal exponential kernel and regression error obtained for each explored combination of data gain g_{data} and t_{refr} (details in 5.2.2.1). Best results in bold.

SETTINGS		**RESULTS**	
gain (dimensionless)	**refractory time (ms)**	**best decay of kernel $\tau_{\text{post}}^{\text{best}}$ (ms)**	**MAE (degrees)**
$1.0 \cdot 10^5$	1.0	500	9.47 ± 2.57
	2.0	500	9.38 ± 2.52
$1.7 \cdot 10^5$	1.0	500	9.04 ± 2.44
	2.0	500	8.95 ± 2.38
$3.0 \cdot 10^5$	1.0	500	$\mathbf{8.84 \pm 2.28}$
	2.0	500	$\mathbf{8.84 \pm 2.26}$
$5.5 \cdot 10^5$	1.0	500	9.02 ± 2.36
	2.0	500	9.06 ± 2.33
$1.0 \cdot 10^6$	1.0	500	9.39 ± 2.53
	2.0	500	9.58 ± 2.62

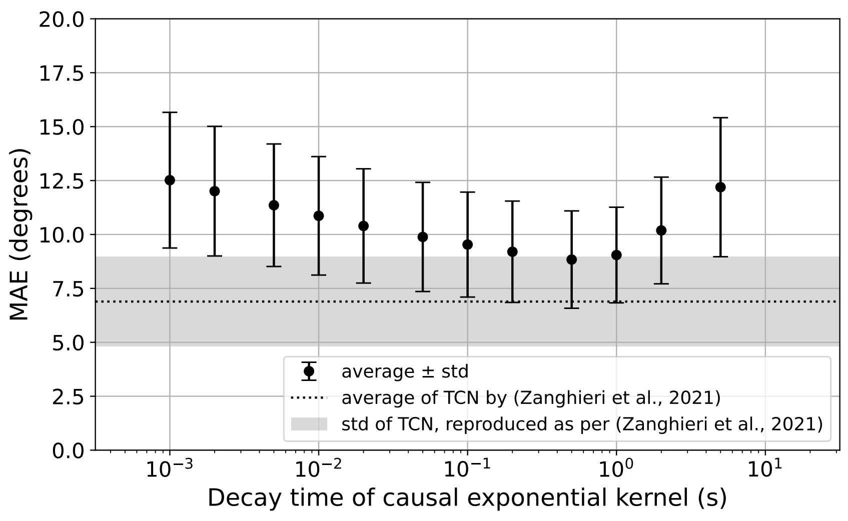

Figure 5.5: Tuning curve of the decay time $\tau_{\text{post}}^{\text{best}}$ of the causal exponential kernel, for the optimal settings data gain $g_{\text{data}} = 3.0 \cdot 10^5$ and refractory time $t_{\text{refr}} = 2\,\text{ms}$.

is shorter than the sampling time of the NinaPro DB8 dataset (namely, $< 500\,\mu\text{s}$) and of most sEMG applications. Thus, the method meets the real-time constraints of an sEMG-based gesture recognition device.

Table 5.4: Profiling of the proposed event-based encoding and regression processing, compared with the SoA TCN setup.

	Platform	**Memory (kB)**	**Operations**	**Latency**		**Power (mW)**	**Energy per inference (µJ)**	**MAE (degrees)**
				cycles (k)	**time (µs)**			$\mu \pm \sigma$
TCN of [24]	GAP8[a] [33]	70.90	$6.32{\cdot}10^6$ in `int32`[b]	476	4760	51.0	243.0	6.89 ± 2.08[c]
This work	STM32 F401RE	7.19	$7.07{\cdot}10^3$ in `fp32`	37.69 ± 0.02	448.6 ± 0.2	40.5	18.2	8.84 ± 2.26

[a] `https://greenwaves-technologies.com/gap8_mcu_ai/`
[b] 8-bit models use 8-bit weights and maps, but layers run in `int32` and outputs are requantized to `uint8` after activation.
[c] The TCN's MAE standard deviation is not reported in [24], so it was determined by reproducing the setup of the original work.

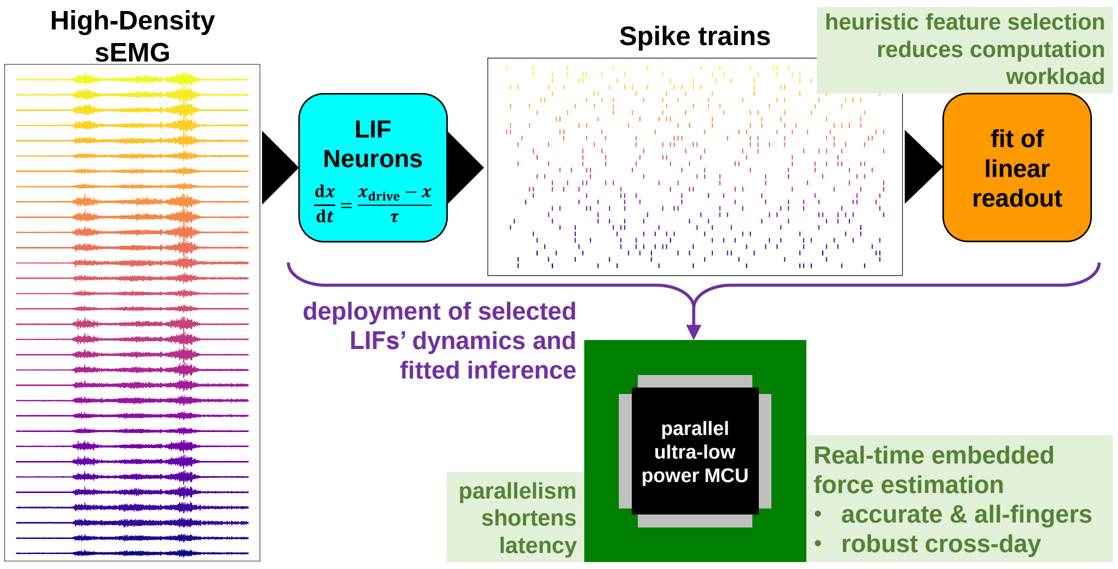
High-Density sEMG
LIF Neurons
$\frac{dx}{dt} = \frac{x_{\text{drive}} - x}{\tau}$
Spike trains
heuristic feature selection reduces computation workload
fit of linear readout
deployment of selected LIFs' dynamics and fitted inference
parallel ultra-low power MCU
parallelism shortens latency
Real-time embedded force estimation
• accurate & all-fingers
• robust cross-day

Figure 5.6: Global scheme of the work of the present section.

5.3 Event-based Estimation of Hand Forces from High-Density Surface EMG on a Parallel Ultra-Low-Power Microcontroller

5.3.1 Overview

This section expands the results of the previous one (5.2), presenting a lean, bio-inspired strategy (Figure 5.6) for an event-based encoding of the sEMG for force estimation, implemented and validated on an ultra-low-power microcontroller suitable for embedded control systems. The approach is motivated by the long-term research interest in validating the accuracy and profiling the execution of event-based techniques for future implementation onto event-based computing platforms as an alternative to the dominant DL models relying on matrix multiplication on temporal data buffers [24], [106]. In contrast, event-based computing promises reduced computation latency and energy consumption [198]–[200].

This contribution presents a technique for encoding the High-Density sEMG (HD-sEMG) signal into an event format that successfully preserves the information content required for the multi-finger force estimation regression task. The contribution is three-fold:

- I present an event-based method that processes the HD-sEMG samples one by one in streaming, updating its state and generating spike trains;

- I tune the parameters of the method on the real HD-sEMG regression dataset HYSER, obtaining a Mean Absolute Error (MAE) of $(8.42 \pm 2.80)\%$ of the Maximum Voluntary Contraction (MVC) in a multi-day, multi-finger scenario, on a par with the literature that addresses easier settings;
- I deploy and profile the setup on a parallel ultra-low power MCU, getting a power consumption $\leq 23.1\,\text{mW}$, an energy draw $\leq 6.37\,\mu\text{J}$ per sample ($2.8\times$ to $11\times$ more energy-efficient than the reference SoA single-core baseline [25], and a latency $\leq 280\,\mu\text{s}$ per sample, shorter than HYSER's HD-sEMG sampling period, thus compatible with real-time processing.

This research is an extension of the previous one (5.2) [25] and expands its heuristic findings as outlined in Table 5.5; namely, here I show that the event-based encoding method remains accurate and versatile if ported from kinematics regression based on sparse sEMG to force estimation based on HD-sEMG, always within strict latency and resource limits. I released open-source the code developed for this research.[4]

5.3.2 Materials & Methods

5.3.2.1 HYSER dataset

This contribution targets the dataset High-densitY Surface Electromyogram Recordings (HYSER)[5] [210], an open-access HD-sEMG dataset realized for research on hand gesture recognition and force estimation. The background on sEMG for ML/DL-driven HMIs is provided in Section 2.4. The dataset was collected from 20 healthy participants, each undergoing two sessions at a distance of 3 to 25 days (8.5 ± 6.7 days on average). The HD-sEMG data were acquired with four 8×8 HD-sEMG arrays (256 channels in total) placed two on each side of the forearm on the extensor and flexor muscles, using an OT Bioelettronica Quattrocento system and sampling at 2048 samples/s. Force signals were acquired during isometric contractions, with a sensor-amplifier pair for each finger, using Huatran SAS sensors and Huatran HSGA amplifiers, sampling at 100 samples/s.

The HYSER dataset is composed of 5 sub-datasets:

1 `PR`: pattern recognition on 34 hand gestures;

2 `MVC`: trials for determining the MVC of every finger's flexion and extension;

3 `1-DoF`: single-finger contractions, for 1-Degree of Freedom (DoF) force estimation;

[4] https://github.com/pulp-bio/hdsemg-force-regression
[5] https://www.physionet.org/content/hd-semg/1.0.0/

Table 5.5: Outline of the present work as an extension of the previous contribution (5.2) [25]. For a fair comparison, this work's profiling shown here refers to 64 Leaky Integrate-&-Fire neurons as 5.2 [25]; the complete profiling results are exposed in 5.3.3.3.

WORK	**Regression target**	**Approach**	**sEMG type: dataset**	**SoA baseline**	**MCU: processor(s)**	**Results**	
						MAE (average $\pm$ std)	**profiling**
Zanghieri et al. [25] (extended here)	kinematics: joint angles	**event-based encoding**	sparse sEMG: NinaPro DB8	[24]	STM32 F401: ARM Cortex-M4F	(8.84 ± 2.28) degrees	latency: 448 µs energy: 18.2 µJ
This work	dynamics: forces		HD-sEMG: HYSER `RANDOM`	[210] [211]–[213]	GWT GAP9: 8 RISC-V cores	$(8.42 \pm 2.80)\%$ MVC	latency: 69.6 µs energy: 1.55 µJ

4 `N-DoF`: multi-finger contraction following prescribed combinations and trajectories, for 5-DoF force estimation in controlled conditions;

5 `RANDOM`: with multi-finger contractions performed in a fashion defined *random task*, i.e., with no prescribed protocol of combinations or trajectories.

Datasets 2 to 5 contain the forces of individual fingers for research on force estimation. In particular, this contribution focuses on the `RANDOM` dataset, which consists of 5 trials per subject, each lasting 25 s, performed with a 5 s inter-trial rest to prevent muscle fatigue.

Most literature on HYSER focuses on discrete gesture recognition on the `PR` dataset, whereas few works to date have addressed continuous force estimation on `1-DoF` and `N-DoF`. Moreover, the `RANDOM` dataset is only dealt with in the basic benchmarking of the first HYSER paper [210]. Table 5.6 reports the SoA works on the HYSER regression datasets. This work is (*i*) the first to tackle HYSER's `RANDOM` dataset in a multi-day setting; (*ii*) the first to deploy and profile the regression algorithm for the HYSER task on a hardware platform suitable for low-power, low-latency wearable HMIs. Thus, this work addresses the working conditions closest to reality, where the force ranges and trajectories are not predefined and can differ from training to test. Moreover, this contribution presents a regressor that is explicitly designed to be hardware-friendly, considering the power, energy, and latency constraints of wearable real-time HMIs.

5.3.2.2 Event-based encoding

The pipeline encodes the raw sEMG to events with a power-based approach inspired by how the mammalian cochlea transduces various frequencies into neural spike trains. The principle of using a bank of filters combined with neural integration has been a model for many designers of bio-inspired hardware to implement circuits that imitate the event encoding happening in nature, either in the digital [205] or analog domain [206], [207].

The method executes the conversion to events by implementing a set of Leaky Integrate-and-Fire (LIF) neurons, each associated with one of the processed sEMG signals. The LIF is a very parsimonious model of the biological neuron, characterized by an inner membrane potential $V_{\mathrm{mem}}(t)$ that follows the electrical law

$$\frac{\mathrm{d}V_{\mathrm{mem}}}{\mathrm{d}t} = -\frac{(V_{\mathrm{mem}} - \mathcal{E}_{\mathrm{leak}}) - \frac{I_{\mathrm{inj}}(t)}{g_{\mathrm{leak}}}}{\tau} \tag{5.16}$$

where τ is the membrane relaxation time, $\mathcal{E}_{\mathrm{leak}}$ is the constant leak reversal potential, I_{inj} is the injected current, and g_{leak} is the constant leak conductance. When V_{mem}

Table 5.6: Overview of the literature of force regression on the HYSER datasets.

WORK	**Interest**	**Approach**	**HYSER sub-dataset**	**Data split**	**Numerical results (average ± std)**	**HW?**
Jiang et al. [210] (2021)	dataset presentation	FIR kernel	`RANDOM`	within-day, leave-1-trial-out	RMSE $= (8.57 \pm 5.27)\%$ MVC	✗
Jiang et al. [211] (2022)	channel selection	FIR kernel + random masks	`N-DoF`	**cross-day**, leave-1-subject-out	RMSE $= (8.66 \pm 0.96)\%$ MVC	✗
Jiang et al. [212] (2023)	robustness vs. noise, physiological explainability	deep forests	`1-DoF`	**cross-day: train on day 1, test on day 2**	RMSE $= (8.0 \pm 2.3)\%$ MVC $r_{\text{Pearson}} = 0.900 \pm 0.101$ $R^2 = 0.631 \pm 0.172$	✗
Wu et al. [213] (2023)	extraction of motor units	gCKC BSS + cumulative spike train + linear regression	`1-DoF`	no ML-style validation	$r_{\text{Pearson}} = 0.908 \pm$ n.a.	✗
This work	event-based ML embedded on parallel ultra-low power MCU	encoding as events + linear regression	`RANDOM`	**cross-day: train on day 1, test on day 2**	MAE $= (8.42 \pm 2.80)\%$ MVC	✓

surpasses a fixed threshold level V_{thr}, the LIF creates a spike, which acts as an emitted event associated with the time of crossing t_{spike}. Then, the LIF is subject to a refractory time t_{refr}, defined as a segment of time $[t_{\text{spike}}, t_{\text{spike}} + t_{\text{refr}}]$ where the LIF is forced to a reset value V_{reset}:

$$V_{\text{mem}}(t) \equiv V_{\text{reset}} \qquad t \in [t_{\text{spike}}, t_{\text{spike}} + t_{\text{refr}}], \tag{5.17}$$

also pausing the response to the inhomogeneous driving term $I_{\text{inj}}(t)/g_{\text{leak}}$.

The numerical LIF emulation for accuracy-oriented regressions does not need to account for the electrical nature of the bio-inspired model. This makes it convenient to change variables to remove the electrical quantities and simplify the notation:

$$x(t) \triangleq \frac{V_{\text{mem}}(t) - \mathcal{E}_{\text{leak}}}{V_{\text{thr}} - \mathcal{E}_{\text{leak}}} \tag{5.18}$$

$$x_{\text{drive}}(t) \triangleq \frac{1}{V_{\text{thr}} - \mathcal{E}_{\text{leak}}} \cdot \frac{I_{\text{inj}}(t)}{g_{\text{leak}}}. \tag{5.19}$$

The physical sense of this transformation is to refer the membrane voltage to the constant $\mathcal{E}_{\text{leak}}$, and measure the membrane voltage and $I_{\text{inj}}(t)/g_{\text{leak}}$ (dimensionally a tension) as a fraction of of $V_{\text{thr}} - \mathcal{E}_{\text{leak}}$, which is the dynamic range of the system. This yields a dimensionless state $x(t)$ with firing threshold $x_{\text{thr}} = 1$ and law

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -\frac{x - x_{\text{drive}}}{\tau}. \tag{5.20}$$

This form of the law makes it more evident that the injected current plays the role of an external driving term. The processing encodes the sEMG as events using each channel to drive an independent LIF unit. The experiments set the relaxation time to $\tau = 10\,\text{ms}$ and the refractory time to $t_{\text{refr}} = 2\,\text{ms}$. To create the driving term from the raw sEMG data, the dataset values sEMG(t) are multiplied by an empirical gain g_{data}, converting the arbitrary-units into an $x_{\text{drive}}(t)$:

$$x_{\text{drive}}(t) = g_{\text{data}} \cdot |\text{sEMG}(t)|, \tag{5.21}$$

experimenting different values of g_{data}. Since the signals of the HYSER dataset are available as voltage values, g_{data} has dimensions V^{-1}.

After each LIF, a post-synaptic potential $x_{\text{post}}(t)$ is simulated, driven by the spikes generated by the corresponding source LIF. This potential also undergoes relaxation, with a relaxation time τ_{post} that causes decay toward 0 if no spikes are received. More formally, x_{post} obeys the law

$$\frac{\mathrm{d}x_{\text{post}}}{\mathrm{d}t} = -\frac{x_{\text{post}}}{\tau_{\text{post}}} + \sum_{t_{\text{spike}}} \delta(t - t_{\text{spike}}) \tag{5.22}$$

where δ denotes the Dirac delta and represents the fact that x_{post} is raised by $+1$ (dimensionless) increments at each received spike. The relaxation regulated by τ_{post} has the effect of a causal exponential decay kernel [208], a form of rate encoding. This rate encoding is equivalent to an event count that, at the present time t, weights every spike in the past as $\exp(-(t - t_{\text{spike}})/\tau_{\text{post}}) \leq 1$. This causal exponential-kernel rate always takes values in the range

$$0 \leq x_{\text{post}} < \frac{1}{1 - e^{-t_{\text{refr}}/\tau_{\text{post}}}} \tag{5.23}$$

In the experiments, τ_{post} is set to 250 ms, thus getting x_{post} values in the range $[0, 125.5)$. Finally, all LIFs' x_{post} values are used as the input regression features for force estimation.

Numerically, the simulations of the LIF neurons can be implemented as discrete updates of x and x_{post}:

$$x \leftarrow x \cdot e^{-\frac{\Delta t}{\tau}} + x_{\text{drive}} \cdot (1 - e^{-\frac{\Delta t}{\tau}}) \tag{5.24}$$

$$x_{\text{post}} \leftarrow x_{\text{post}} \cdot e^{-\frac{\Delta t}{\tau_{\text{post}}}} + 1\,[\text{if spike}] \tag{5.25}$$

where Δt is the discrete time step of the simulation; Equation 5.24 is skipped during refractories. A key feature of Equations 5.24) and 5.25) is that they update *online*, i.e., they consume one single sEMG input for each channel at a time, computing x_{drive} and then the new x_{pre} and x_{post}; at the next sampling period, the new sEMG input data overwrite the old ones. Hence, the size of the input data stored at each sampling period never exceeds $N_{\text{ch}} \times 4\,\text{bytes} = 256 \times 4\,\text{bytes} = 1\,\text{KiB}$ for `float32` data on the $N_{\text{ch}} = 256$-channel HYSER dataset. In addition, L_1-regularization is applied for a data-driven channel selection, further reducing the size of inputs and computation, as explained in 5.3.2.3.

Two implementations of the event-based encoding and the inference were developed:

- in Python (v. 3.8), I directly configured the `NeuronGroup` and `Synapses` classes native to the simulator Brian2 [209] v. 2.5;
- in C, for deployment on the MCU, I implemented (5.24) and (5.25) and refractories, parallelizing as detailed in 5.3.2.4.

The Python and C implementations were used to compute the regression error statistics in offline experiments on a PC and online experiments on GAP9, respectively. In the Python offline experiments, the whole output time series of each HYSER `RANDOM`'s 25-second recording is available for taking the error statistics. In the online experiments, a PC sends the HYSER's samples to the MCU via a serial interface in streaming; the MCU consumes them to update the neurons' states and the inference; finally, the MCU sends the inference output back to the PC. The GAP9 MCU repeats the reception-processing-transmission loop online for each sample of every HYSER `RANDOM`'s 25-second recording. On the PC side, a script reads the regression outputs and saves them into an array. This series is used to check the numerical match of the results of the Python and the C implementations and to compute the regression error statistics.

5.3.2.3 Regression

Force estimation is tackled with linear regression to ensure reduced memory and computation requirements, prioritizing the low embedded resource budget. With the $N_{\text{ch}} = 256$ HD-sEMG channels and the $N_{\text{DoFs}} = 5$ of the HYSER dataset, the force estimation task is framed as a

$$\mathbf{x} \in \mathbb{R}^{N_{\text{ch}}} \longmapsto \mathbf{y} \in \mathbb{R}^{N_{\text{DoFs}}} \tag{5.26}$$

multivariate, multi-target regression, parameterized by a $\mathbf{W} \in \mathbb{R}^{N_{\text{ch}} \times N_{\text{DoF}}} = \mathbb{R}^{5\times 256}$ coefficients matrix and an intercept $\overline{\mathbf{y}}_{\text{train}} \in \mathbb{R}^{N_{\text{DoFs}}} = \mathbb{R}^5$ equal to the sample mean of the training set values. In `float32` format, each inference amounts to a memory footprint of 6184 bytes (including input and output) and 1285 FLOP.

To keep the processing hardware-friendly, the resource budget is reduced by applying a strong L_1 regularization. The target function of the regression is thus

$$\frac{1}{N_{\text{train}}} \sum_{i}^{N_{\text{train}}} \left\|\hat{\mathbf{y}}_i - \mathbf{y}_i\right\|_2^2 + \alpha \left\|\mathbf{W}\right\|_1 \tag{5.27}$$

where $\mathbf{y}_i, \hat{\mathbf{y}}_i = \mathbf{W}\mathbf{x} \in \mathbb{R}^5$ are the multivariate ground truth and estimation, respectively, corresponding to the i-th inference, N_{train} is the total number of inferences (equivalent to the training set size, i.e., each HYSER `RANDOM` Day 1 session), $\|\cdot\|_2$ is the Euclidean norm, $\|\cdot\|_1$ is the L_1-norm, and α is the parameter governing the amount of regularization (notice that it does not get divided by N_{train}). The amount of regularization is tuned by exploring different values for α.

In addition to countering overfitting, the prerogative of the L_1 regularization is to perform automatic sEMG channel selection. The reason why L_1 regularization results in automatic feature selection is that it induces sparsity since reducing any coefficient benefits the penalty term equally, regardless of the coefficient's magnitude; in contrast, $L_{p>1}$-norms privilege reducing the larger coefficients (since the $p > 1$ exponent makes their contribution to the norm larger), thus making $L_{p>1}$-norms less likely to push coefficients to 0.

The L_1-induced data-driven feature selection on the sEMG channels results in fewer associated LIF units compared to HYSER's $N_{\text{ch}} = 256$ total sensors. Channel selection reduces the application's requirements, namely input data bandwidth, memory footprint, and computational load. This reduction makes the processing more hardware-friendly for resource-constrained computation devices such as the MCU targeted in this work (5.3.2.4). Moreover, this data-driven reduction experimentally determines the number of channels actually required for an accurate regression. So, in this setup, L_1-regularization is the key for studying the integration of HD-sEMG acquisition setup and embedded platforms.

As a dataset split on HYSER `RANDOM`, the Day 1 session was used for training and the Day 2 session was used for validation; for both sessions, all the 5 trials were used. Training and test were run separately for every subject without any multi-subject training or inter-subject validation. Together with the choice of HYSER `RANDOM` itself, this dataset split is the most challenging and closest to a real test scenario; no previous work on HYSER has addressed multi-day inference of 5-finger forces (Table 5.6).

The regression accuracy is determined using the Mean Absolute Error (MAE):

$$\text{MAE} = \frac{1}{N_{\text{infer}} N_{\text{DoF}}} \sum_{i=1}^{N_{\text{infer}}} \|\hat{\mathbf{y}}_i - \mathbf{y}_i\|_1 \tag{5.28}$$

where N_{infer} is the total number of inferences, i.e., the validation set size, and the rest follows the notation of 5.3.2. The error is measured as a fraction of the MVC, as is the standard approach [210]–[213]. To rescale forces to the MVC scale, the MVC was determined for each direction (i.e., flexion or extension) of each finger of each subject using the data from HYSER `MVC` following the same heuristic as suggested by the dataset authors, namely determining the MVC as the average of the 200 strongest values.[6] Assessing the control quality via the MAE is convenient because the MAE is first-order, thus more robust to outliers than quadratic statistics such as the (R)MSE or the multivariate coefficient of determination R^2. The MAE is averaged over time (i.e., over the 5 trials of each session) and over all 5 finger DoFs, as expressed by (5.28), and over all 20 participants.

5.3.2.4 Deployment and profiling on a parallel ULP MCU

The numerical model was deployed of the LIF neurons associated with each HD-sEMG onto the commercial MCU GAP9 (2.2; Figure 4.15), which features a Parallel

[6] `https://www.physionet.org/content/hd-semg/1.0.0/toolbox/function`

Ultra-Low Power (PULP) [62], [63] 9-core cluster accelerator based on the RISC-V Instruction Set Architecture extended with specialized DSP and ML instructions. This device is a SoA low-power processor that ranked first in latency and energy consumption on the benchmarks MLPerf Tiny v1.0 [39]. In a potential complete prototype implementing the HD-sEMG-based control policy, the MCU's role is computation, i.e., the data processing that consists of updating the LIF neurons' states and executing the regression inference. In contrast, the upstream functions of data sampling and transmission are to be performed by other ADC and interface modules that produce the data and convey them to the MCU.

GAP9 was programmed implementing the LIF update steps in C with parallelization. Parallelization comes naturally since all LIFs are independent as to both state variables and operations, so LIFs were distributed evenly across the cores, with a workload difference of 1 LIF at maximum in the experiments where LIFs are not a multiple of the used cores. A core getting assigned a LIF means that the core will execute all that LIF's x and x_{post} discrete update iteration steps. The implementation parallelizes over up to 8 cores since GAP9's cluster's ninth core, referred to as Master Core or Core 8, only serves as a cluster controller and manages Direct Memory Access (DMA) memory transfers.

For profiling, the configuration used the settings corresponding to GAP9's highest energy-efficiency, namely $V_{\mathrm{dd\ core}} = 0.65\,\mathrm{V}$ and $f_{\mathrm{CLK}} = 240\,\mathrm{MHz}$. Latency was measured in cycles by exploiting the performance counter exposed by the API of PMSIS,[7] the open-source system layer for GAP9's operating system. Latency in physical time was determined as num_cycles/f_{CLK}. The power draw was measured experimentally, using the GAP9's Evaluation Kit[8,9] and a Nordic Semiconductor Power Profiler Kit II (PPK2).[10] The PPK2 measured the current consumption of GAP9's core, excluding the peripherals and the off-chip memories. A GPIO was used to synchronize the current measurement with the code execution. Finally, the energy consumption was determined as power$\times$latency.

5.3.3 Experimental Results

5.3.3.1 Time-domain behavior

Figure 5.7 displays a representative example of the time-domain behavior of the finger force estimation provided by the algorithm. The reported results are from HYSER's

[7] `https://greenwaves-technologies.com/manuals/BUILD/HOME/html/index.html`
[8] `https://greenwaves-technologies.com/product/gap9_evk-gap9-evaluation-kit-efused/`
[9] `https://greenwaves-technologies.com/product/gap9-resources/`
[10] `https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit-2`

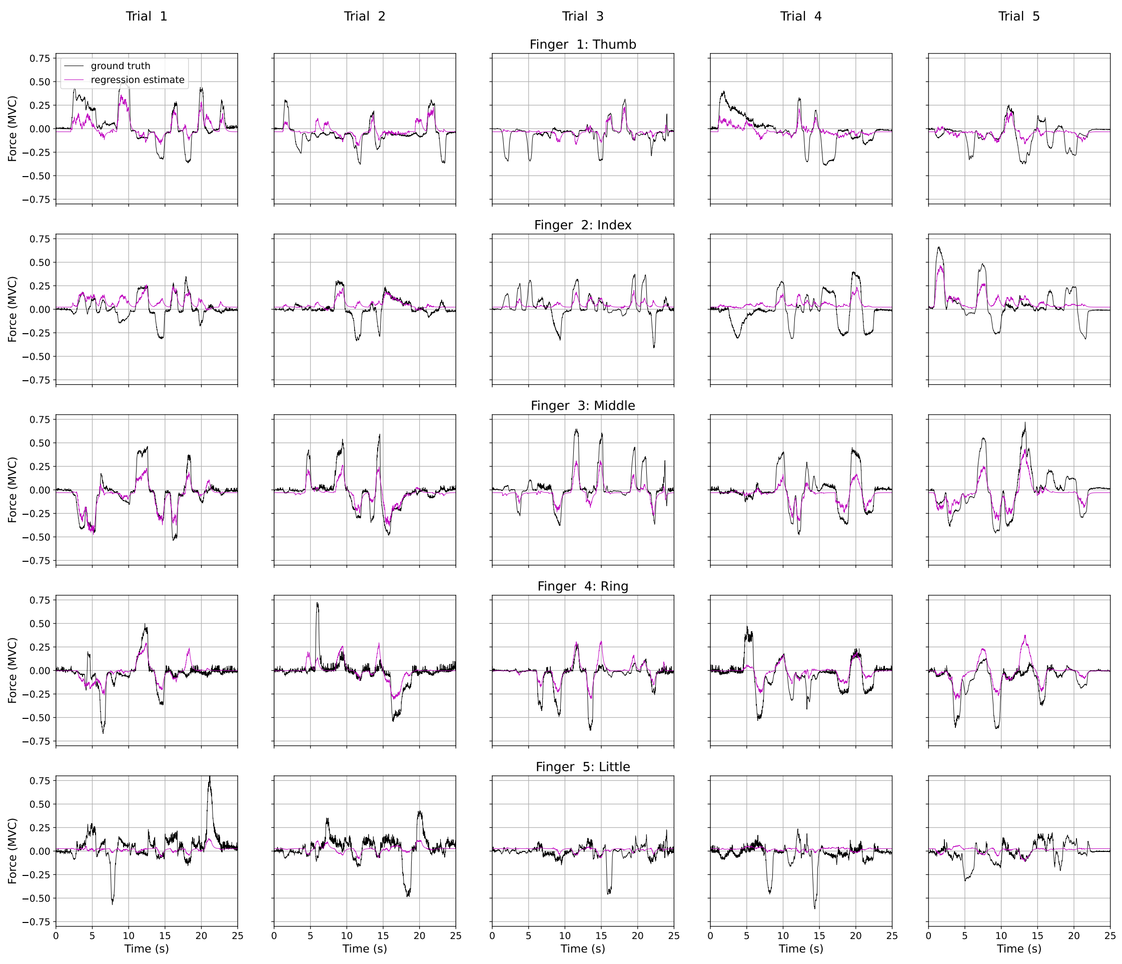

Figure 5.7: Force estimation results obtained for regression inference on HYSER's Subject 1, `RANDOM` dataset, Day 2 (the one not seen in training), all 5 trials, all 5 fingers.

Subject 1, `RANDOM` dataset, Day 2 (i.e., the one never seen in training), all 5 trials, all 5 fingers. These data are chosen for display as they are representative of the general time-domain trends observed in the results.

The reported trials contain many examples of good regression quality, especially for the thumb, middle, and ring fingers. The rest position is generally well-modeled, with a good match between the ground truth and the estimation when force is in the interval ± 0.05 MVC. The timing of the falling and rising fronts, corresponding to the dynamic phases of flexions and extensions, respectively, are also accurate. In contrast, the estimation errors mainly happen in the central regions of flexions and extensions, where the estimation often stops before the force reaches its full amplitude; this happens both in some steady central segments and in some triangular peaks. Interestingly, this kind of time-domain behavior, with accurate timing in transients and a central offset error, is the same as observed in the estimation of the hand kinematics (5.1) [24].

The displayed results also contain the typical estimation errors obtained with the algorithm, especially in the index finger trials. A common regressor's mistake is failing to recognize negative forces, i.e., finger flexions. This error is one of the most frequent erratic behaviors in the results. In future work, the trials with poor modeling of contractions can be addressed by adding a flexion-vs-extension(-vs-rest) detector before the regressor, using the regressor only for estimating the amount of force and the dedicated detector for recognizing the force's sign. The little finger's results are the ones showing the least accurate regression. The reason why this poor modeling does not harm the overall performance of the method is that the little finger has an MVC, and thus a force dynamic range, that is on average $(0.55 \pm 0.07)\times$ compared to the other single fingers. This means that the little finger contributes to the global dynamics of the hand forces by approximately one-half compared to the other single fingers, making errors less impactful from an end-to-end application viewpoint.

It is worth remarking that the regression issues illustrated here are only discussed to present an overview of the most typical errors from a time-domain point of view. These behaviors do not compromise the average regression quality. The overall competitiveness of the method compared to the SoA is shown by the regression error statistics presented in the next subsection.

5.3.3.2 Regression error

Figure 5.8 shows the regression error obtained evaluating a grid of pairs of g_{data} and α, exploring

$$g_{\text{data}}\left[\mathrm{V}^{-1}\right] = \quad 5.0,\ \ 10.0,\ \ 15.0,\ \ 20.0,\ \ 25.0,\ \ 30.0 \tag{5.29}$$

$$\alpha = \quad 10^{-1},\ \ 10^{-1.5},\ \ 10^{-2}. \tag{5.30}$$

The fits with the mildest regularization $\alpha = 0.01$ do not yield reliable results: the high variability of the MAE means a high regression error for a relevant fraction of fingers or subjects; $\alpha = 0.01$ proves thus poor methodologically. Pushing regularization to $\alpha = 0.032$ makes results more stable, improving both the MAE's average and variance for all values g_{data}. Tightening to $\alpha = 0.1$ further improves both averages and variances for all the explored g_{data} yielding the lowest error, i.e. $(8.42 \pm 2.80)\%$ MVC for $g_{\text{data}} = 15.0\,\mathrm{V}^{-1}$. These results prove that the method can work in a multi-day multi-finger setup in the absence of fixed force exercises, achieving results in the same range as previous works that tackled the HYSER dataset in easier settings, namely within-day [210], with predefined force protocol [211], or single-finger [212] (as summarized in Table 5.6), whereas the validation is closer to actual non-laboratory scenarios.

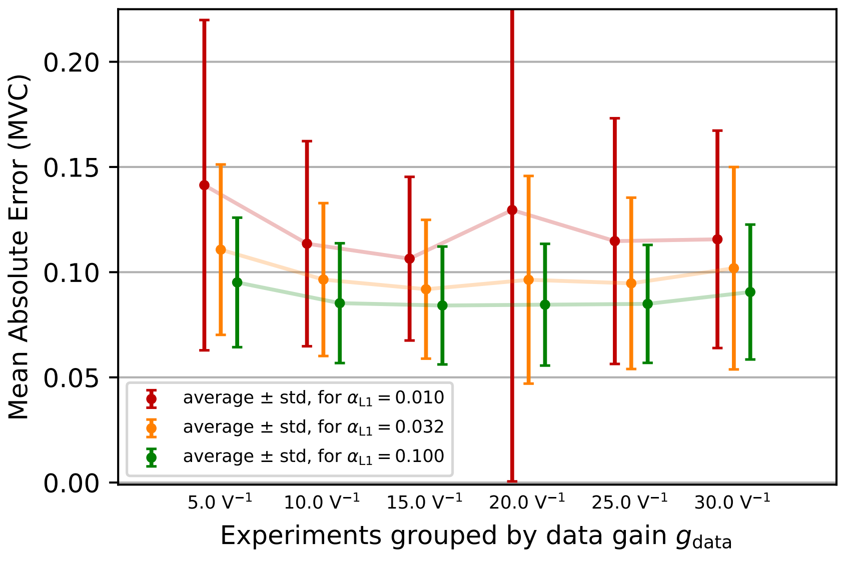

Figure 5.8: Regression error results. Explored data gains shown as categorical to allow horizontal x-displacement to de-overlap bars.

In the perspective of pursuing real-world implementations, error variability is methodologically as essential as error average to ensure that a method is capable of uniform performance across different users. The standard deviation is due to the inherent variability across subjects and sessions, which produce sEMG data with easier or harder patterns. Tightening the screw of regularization has decreased both the average error and its dispersion, proving more beneficial than adjusting g_{data}, which only yielded plateaus with uniform variance.

As to the feature sparsity obtained from the L_1 regularization, the identified optimal solution $\alpha = 0.1$, $g_{\text{data}} = 15.0\,\text{V}^{-1}$ has a minimum of 42 sEMG channels (out of $N_{\text{ch}} = 256$) with a non-zero coefficient for at least one of the 5 fingers (i.e., maximum sparsity of 83.6%, for subject 6), and a maximum of 84 (i.e., minimum sparsity 67.2%, for subject 13), with a median of 55 (sparsity 78.5%, for subjects 10 and 18). These results are not only competitive but also interesting for the insight they provide in the perspective of integrating HD-sEMG with embedded systems. On the one hand, the heuristic range 42 – 84 is above the typical channel count of a low-density, sparse sEMG setup, confirming that the use of HD-sEMG is motivated. On the other hand, the resulting 42 – 84 channels are much fewer than HYSER's total $N_{\text{ch}} = 256$ (i.e., sparsity is high), informing in a data-driven way that an accuracy-oriented application does not require channel counts of the order of 100 or even 200. So, a key insight of the regression results is the empirical estimate of the useful channel count actually needed for a low regression error.

The amount of L_1-induced data-driven sparsity also shapes the conclusions of the on-device profiling results (5.3.3.3) since the application only needs to implement the LIFs associated with the selected input channels. For instance, a consequence of sparsity is on input data memory footprint: sparsity lowers the size of input data from the theoretical maximum of 1 MiB (as explained in 5.3.2.2) to a minimum of 168 bytes, a maximum of 336 bytes, and a median of 220 bytes across subjects, corresponding to the values of 42, 84, and 55 channels reported above. These results prove that L_1-regularization contributes to making the method hardware-friendly in the presence of a high number of input channels, such as the 256 sensors of the HYSER dataset.

5.3.3.3 Profiling

Figures 5.9 and 5.10 show the profiling results regarding speedup and latency on 8 cores, respectively.

The experimental speedup on 8 cores (Figure 5.9) is close to 8× and is better for a higher number of simulated LIF neurons. The speedup on 8 cores for 64 and 256 LIFs is 7.45× and 7.81× on a theoretical maximum of 8×. Since all LIFs are independent, the algorithm is fully parallelizable mathematically. The only sequential part when executing on HW is the initial DMA transfer of one `float32` value per channel from the MCU's L2 memory to the cluster's L1 memory for faster access during the subsequent computation. This DMA transfer takes < 2 cycles per `float32` datum. According to Amdahl's law, this overhead yields an ideal speedup of 7.95× on 8 cores, equal for all the profiled workloads since both the transfer and the computation are proportional to the number of LIFs. Considering the Amdahl upper bound, the obtained speedup is 97.6% and 98.3% of the Amdahl ideal for 64 and 256 LIFs, respectively.

The latency results (Figure 5.10) show that the implementation satisfies the real-time constraint since it can update all LIFs within the sampling period

$$T_{\text{sEMG HYSER}} = \frac{1}{2048\,\text{Hz}} \approx 488\,\mu\text{s}, \tag{5.31}$$

thus proving able to use every sEMG sample as a driving term for the corresponding LIF. The highest workload case, i.e., 256 LIFs, has a latency of 276 µs. We observe that workloads between 64 and 96 LIFs have a latency of 69.6 µs to 103 µs, which is in the range of 1/7 to 1/5 of the available time. This range of workloads is similar to the range of lower-sparsity subjects identified in the regression results of the previous subsection (5.3.3.2), namely 55 (median) to 84 (maximum) LIFs. These results confirm that the amount of parallelism pursued is required by the use case, motivating the choice of the parallel platform. Overall, the solution is consistent with the HYSER dataset and with SoA sEMG applications, characterized by sampling frequencies typically $> 1\,\text{kHz}$.

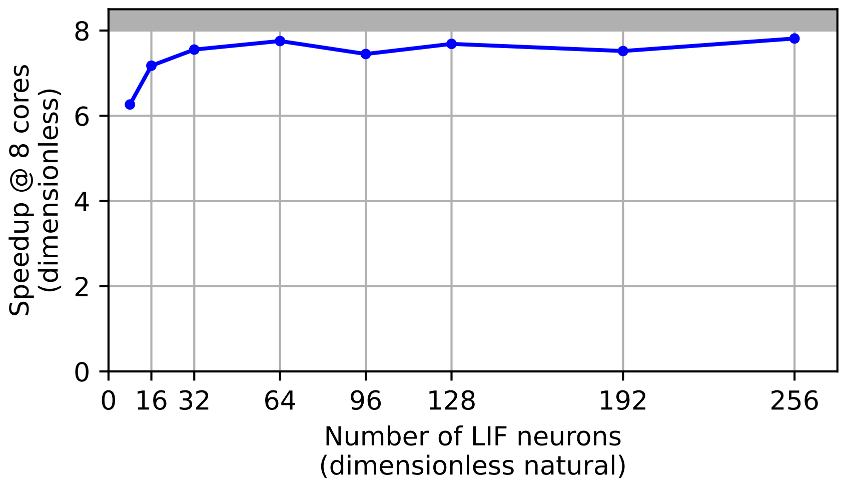

Figure 5.9: Speedup on 8 cores obtained for different numbers of executed LIF neurons. The grey region is the unreachable speedup $\geq 8\times$.

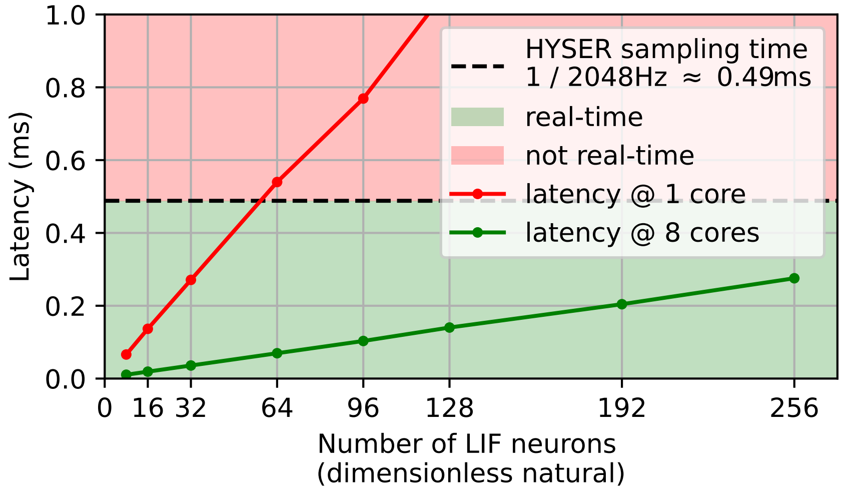

Figure 5.10: Latency as a function of the number of executed LIF neurons, on 8 cores.

The measured power consumption is 22.3 mW and 23.1 mW for 64 and 256 LIF neurons respectively, with a respective energy consumption of 1.55 µJ and 6.35 µJ. In particular, the energy draw for 64 LIFs is 11.7× lower compared to the 18.2 µJ of the previous work with 64 LIFs deployed on a single-core MCU (STM32 F401, featuring ARM Cortex-M4F) [25], denoting the improved energy-efficiency of the parallel implementation.

Chapter 6

Conclusion

This Ph.D. thesis has presented methods and contributions on the topic of energy-efficient time-series analysis addressed with automated learning ported on resource-constrained, low-power computing platforms, with solutions based on DL and traditional non-deep ML. This dissertation has covered different fields in both basic and applied research, ranging from research on the accuracy-efficiency algorithmic tradeoff for biosignal analysis to tasks inspired by industrial use cases. The unifying approach that embraces the whole of this research work is the methodology and interest in addressing time-series ML/DL as a computational task to be executed obeying the resource constraints characteristic of low-power embedded computing platforms. This perspective has allowed exploiting the energy-efficiency of both single- and multi-core embedded MCUs.

This thesis has spanned the three types of automated learning tasks: binary classification, multi-class (single-label) classification, and regression. Regarding binary classification, I presented a proximity sensor for safety around industrial machinery that is accurate and robust against acoustic noise, and an embedded solution for epilepsy detection from iEEG data. Both setups rely on a TCN deployed onto an edge MCU, demonstrating the methodology's performance and accuracy. As to multi-class (single-label) classification and regression, the research focused on hand modeling from sEMG signals. After developing a non-deployed TCN for discrete hand gesture classification, I advanced the setup by deploying the model onto a multi-core MCU and investigating heuristic unsupervised adaptation to different arm postures. As to regression, needed for continuous and more natural control of HMIs, I presented an embedded TCN for accurate hand kinematics estimation; then, I targeted the modeling of both finger force and hand kinematics using event-based features, which are computationally cheaper and are thus amenable for a future porting onto event-driven devices with reduced latency and energy consumption. The contributions to the topic of sEMG-based hand modeling are fruitful for progress in intuitive and non-invasive HMIs.

It is interesting to highlight that a challenge common to most of the heuristics and analyses presented in this thesis is the generalization on data coming from conditions unseen at training time: iEEG from unseen epileptic seizures, sEMG from unseen acquisition sessions, sEMG from unseen arm postures, and HD-sEMG from unseen force patterns and ranges. It is worth remarking that this success has involved models from both classical, non-deep Machine Learning and Deep Learning. Generalization was mainly achieved thanks to two strategies: (*i*) training on data collected from diverse acquisition settings and (*ii*) designing lightweight algorithms whose compact size induces regularization. The latter constraint coincides with the model compactness desired for embedded deployment and is a general feature of TinyML.

In conclusion, the global contribution of the research work reported in this thesis is the success of the embedded methodology for time-series ML/DL, which has yielded SoA accuracy and efficiency, proving this approach effective and promising for several commercial, clinical, and industrial scenarios.

Bibliography

[1] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: A review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, Mar. 2019. DOI: 10.1007/s10618-019-00619-1.

[2] A. Rajkomar, E. Oren, K. Chen, A. M. Dai, N. Hajaj, M. Hardt, P. J. Liu, X. Liu, J. Marcus, M. Sun, P. Sundberg, H. Yee, K. Zhang, Y. Zhang, G. Flores, G. E. Duggan, J. Irvine, Q. Le, K. Litsch, A. Mossin, J. Tansuwan, D. Wang, J. Wexler, J. Wilson, D. Ludwig, S. L. Volchenboum, K. Chou, M. Pearson, S. Madabushi, N. H. Shah, A. J. Butte, M. D. Howell, C. Cui, G. S. Corrado, and J. Dean, "Scalable and accurate deep learning with electronic health records," *NPJ Digital Medicine*, vol. 1, no. 1, May 2018. DOI: 10.1038/s41746-018-0029-1.

[3] M. Zanghieri, S. Benatti, A. Burrello, V. J. Kartsch Morinigo, F. Conti, and L. Benini, "Robust real-time embedded EMG recognition framework using temporal convolutional networks on a multicore IoT processor," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 2, pp. 244–256, 2020. DOI: 10.1109/TBCAS.2019.2959160.

[4] N. D. Truong, A. D. Nguyen, L. Kuhlmann, M. R. Bonyadi, J. Yang, S. Ippolito, and O. Kavehei, "Convolutional neural networks for seizure prediction using intracranial and scalp electroencephalogram," *Neural Networks*, vol. 105, pp. 104–111, 2018. DOI: 10.1016/j.neunet.2018.04.018.

[5] A. Burrello, F. B. Morghet, M. Scherer, S. Benatti, L. Benini, E. Macii, M. Poncino, and D. J. Pagliari, "Bioformers: Embedding transformers for ultra-low power sEMG-based gesture recognition," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 1443–1448. DOI: 10.23919/DATE54114.2022.9774639.

[6] A. Burrello, D. J. Pagliari, M. Risso, S. Benatti, E. Macii, L. Benini, and M. Poncino, "Q-PPG: Energy-efficient PPG-based heart rate monitoring on wearable devices," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 15, no. 6, pp. 1196–1209, 2021. DOI: 10.1109/TBCAS.2021.3122017.

[7] H. F. Nweke, Y. W. Teh, M. A. Al-garadi, and U. R. Alo, "Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges," *Expert Systems with Applications*, vol. 105, pp. 233–261, 2018. DOI: 10.1016/j.eswa.2018.03.056.

[8] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognition Letters*, vol. 119, pp. 3–11, 2019. DOI: 10.1016/j.patrec.2018.02.010.

[9] W. He, P. Motlicek, and J.-M. Odobez, "Deep neural networks for multiple speaker detection and localization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 74–79. DOI: 10.1109/ICRA.2018.8461267.

[10] M. Azimi, A. D. Eslamlou, and G. Pekcan, "Data-driven structural health monitoring and damage detection through deep learning: State-of-the-art review," *Sensors*, vol. 20, no. 10, 2020. DOI: 10.3390/s20102778.

[11] T. Cerquitelli, D. J. Pagliari, A. Calimera, L. Bottaccioli, E. Patti, A. Acquaviva, and M. Poncino, "Manufacturing as a data-driven practice: Methodologies, technologies, and tools," *Proceedings of the IEEE*, vol. 109, no. 4, pp. 399–422, 2021. DOI: 10.1109/JPROC.2021.3056006.

[12] C. W. Tan, C. Bergmeir, F. Petitjean, and G. I. Webb, "Time series extrinsic regression: Predicting numeric values from time series data," *Data Mining and Knowledge Discovery*, vol. 35, pp. 1032–1060, Mar. 2021. DOI: 10.1007/s10618-021-00745-9.

[13] C. W. Tan, C. Bergmeir, F. Petitjean, and G. I. Webb, *Monash University, UEA, UCR time series extrinsic regression archive*, 2020. DOI: 10.48550/arXiv.2006.10996.

[14] Z. C. Lipton, J. Berkowitz, and C. Elkan, *A critical review of recurrent neural networks for sequence learning*, 2015. DOI: 10.48550/arXiv.1506.00019.

[15] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *CoRR*, vol. abs/1803.01271, 2018. DOI: 10.48550/arXiv.1803.01271.

[16] A. Burrello, A. Dequino, D. J. Pagliari, F. Conti, M. Zanghieri, E. Macii, L. Benini, and M. Poncino, "TCN mapping optimization for ultra-low power time-series edge inference," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '21, IEEE Press, 2021. DOI: 10.1109/ISLPED52811.2021.9502494.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, Curran Associates, Inc., 2017. DOI: 10.48550/arXiv.1706.03762.

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of deep bidirectional transformers for language understanding*, Jun. 2019. DOI: 10.18653/v1/N19-1423.

[19] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20, Red Hook, NY, USA: Curran Associates Inc., 2020. DOI: 10.5555/3495724.3495883.

[20] M. Zanghieri, F. Indirli, A. Latella, G. M. Puglia, F. Tecce, F. Papariello, G. Urlini, L. Benini, and F. Conti, "An extreme-edge TCN-based low-latency collision-avoidance safety system for industrial machinery," *IEEE Access*, pp. 1–1, 2024. DOI: 10.1109/ACCESS.2024.3357510.

[21] M. Zanghieri, A. Burrello, S. Benatti, K. Schindler, and L. Benini, "Low-latency detection of epileptic seizures from iEEG with temporal convolutional networks on a low-power parallel MCU," in *2021 IEEE Sensors Applications Symposium (SAS)*, 2021, pp. 1–6. DOI: 10.1109/SAS51076.2021.9530181.

[22] M. Zanghieri, S. Benatti, F. Conti, A. Burrello, and L. Benini, "Temporal variability analysis in sEMG hand grasp recognition using temporal convolutional networks," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2020, pp. 228–232. DOI: 10.1109/AICAS48895.2020.9073888.

[23] M. Zanghieri, M. Orlandi, E. Donati, E. Gruppioni, L. Benini, and S. Benatti, "Online unsupervised arm posture adaptation for sEMG-based gesture recognition on a parallel ultra-low-power microcontroller," in *2023 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2023, pp. 1–5. DOI: 10.1109/BioCAS58349.2023.10388902.

[24] M. Zanghieri, S. Benatti, A. Burrello, V. J. Kartsch Morinigo, R. Meattini, G. Palli, C. Melchiorri, and L. Benini, "sEMG-based regression of hand kinematics with temporal convolutional networks on a low-power edge microcontroller," in *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, 2021, pp. 1–6. DOI: 10.1109/COINS51742.2021.9524188.

[25] M. Zanghieri, S. Benatti, L. Benini, and E. Donati, "Event-based low-power and low-latency regression method for hand kinematics from surface EMG," in *2023 9th International Workshop on Advances in Sensors and Interfaces (IWASI)*, 2023, pp. 293–298. DOI: 10.1109/IWASI58316.2023.10164372.

[26] M. Zanghieri, P. M. Rapa, M. Orlandi, E. Donati, L. Benini, and S. Benatti, "Event-based estimation of hand forces from high-density surface EMG on a parallel ultra-low-power microcontroller," *IEEE Sensors Journal*, pp. 1–1, 2024. DOI: 10.1109/JSEN.2024.3359917.

[27] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1003–1012. DOI: 10.1109/CVPR.2017.113.

[28] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, *WaveNet: A generative model for raw audio*, 2016. DOI: 10.48550/arXiv.1609.03499.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[30] Arm Holdings, *Cortex-M7*, available at https://developer.arm.com/Processors/Cortex-M7 (24/01/2024).

[31] STMicroelectronics, *STM32H742xI/G STM32H743xI/G*, available at https://www.st.com/resource/en/datasheet/stm32h743vi.pdf (25/01/2024), 2023.

[32] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "Mr. Wolf: An energy-precision scalable parallel ultra low power SoC for IoT edge processing," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, 2019. DOI: 10.1109/JSSC.2019.2912307.

[33] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "GAP-8: A RISC-V SoC for AI at the edge of the IoT," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2018, pp. 1–4. DOI: 10.1109/ASAP.2018.8445101.

[34] GreenWaves Technologies, *Low-power processors*, available at https://greenwaves-technologies.com/low-power-processor/ (25/01/2024).

[35] A. Burrello, "Optimizing AI at the edge: From network topology design to MCU deployment," Ph.D. dissertation, University of Bologna, Bologna, Italy, available at https://amsdottorato.unibo.it/10542/ (25/01/2024), 2023.

[36] A. Rahimi, I. Loi, M. R. Kakoee, and L. Benini, "A fully-synthesizable single-cycle interconnection network for shared-L1 processor clusters," in *2011 Design, Automation & Test in Europe*, 2011, pp. 1–6. DOI: 10.1109/DATE.2011.5763085.

[37] D. Rossi, I. Loi, G. Haugou, and L. Benini, "Ultra-low-latency lightweight DMA for tightly coupled multi-core clusters," in *Proceedings of the 11th ACM Conference on Computing Frontiers*, ser. CF '14, Association for Computing Machinery, 2014. DOI: 10.1145/2597917.2597922.

[38] A. Pullini, D. Rossi, G. Haugou, and L. Benini, "µDMA: An autonomous I/O subsystem for IoT end-nodes," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2017, pp. 1–8. DOI: 10.1109/PATMOS.2017.8106971.

[39] MLCommons, *MLPerf inference: Tiny benchmark suite results*, available at https://mlcommons.org/benchmarks/inference-tiny/ (25/01/2024).

[40] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021. DOI: 10.1016/j.neucom.2021.07.045.

[41] A. Burrello, F. Conti, A. Garofalo, D. Rossi, and L. Benini, "DORY: Lightweight memory hierarchy management for deep NN inference on IoT endnodes: Work-in-progress," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis Companion*, ser. CODES/ISSS '19, Association for Computing Machinery, 2019. DOI: 10.1145/3349567.3351726.

[42] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, and F. Conti, "DORY: Automatic end-to-end deployment of real-world DNNs on low-cost IoT MCUs," *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1253–1268, 2021. DOI: 10.1109/TC.2021.3066883.

[43] F. Conti, *Technical report: NEMO DNN quantization for deployment model*, 2020. DOI: 10.48550/arXiv.2004.05930.

[44] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Comput. Surv.*, vol. 54, no. 8, Oct. 2021. DOI: 10.1145/3469029.

[45] Google DeepMind, *TensorFlow Lite*, available at https://www.tensorflow.org/lite/guide (25/01/2024).

[46] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous distributed systems*, 2016. DOI: 10.48550/arXiv.1603.04467.

[47] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16, USENIX Association, 2016, pp. 265–283. DOI: 10.48550/arXiv.1605.08695.

[48] Google DeepMind, *TensorFlow*, available at https://www.tensorflow.org/ (25/01/2024).

[49] Google DeepMind, *Keras: The high-level API for TensorFlow*, available at https://www.tensorflow.org/guide/keras (25/01/2024).

[50] Google DeepMind, *TensorFlow Lite for microcontrollers*, available at https://www.tensorflow.org/lite/microcontrollers (25/01/2024).

[51] *Open Neural Network Exchange*, available at https://github.com/onnx (25/01/2024).

[52] T. Jin, G.-T. Bercea, T. D. Le, T. Chen, G. Su, H. Imai, Y. Negishi, A. Leu, K. O'Brien, K. Kawachiya, and A. E. Eichenberger, *Compiling ONNX neural network models using MLIR*, 2020. DOI: 10.48550/arXiv.2008.08272.

[53] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, *TVM: An automated end-to-end optimizing compiler for deep learning*, 2018. DOI: 10.48550/arXiv.1802.04799.

[54] *TVM: Open deep learning compiler stack*, available at https://github.com/apache/tvm/tree/main (25/01/2024).

[55] *microTVM design document*, available at https://tvm.apache.org/docs/arch/microtvm_design.html (25/01/2024).

[56] STMicroelectronics, *X-CUBE-AI documentation*, available at https://wiki.stmicroelectronics.cn/stm32mcu/wiki/AI:X-CUBE-AI_documentation (25/01/2024).

[57] STMicroelectronics, *STM32CubeMX*, available at https://www.st.com/content/st_com/en/stm32cubemx.html (25/01/2024).

[58] Arm Holdings, *Common Microcontroller Software Interface Standard (CMSIS)*, available at https://www.arm.com/technologies/cmsis (24/01/2024).

[59] C. N. Coelho, A. Kuusela, S. Li, H. Zhuang, J. Ngadiuba, T. K. Aarrestad, V. Loncar, M. Pierini, A. A. Pol, and S. Summers, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *Nature Machine Intelligence*, vol. 3, no. 8, pp. 675–686, Jun. 2021. DOI: 10.1038/s42256-021-00356-5.

[60] L. Geiger and P. Team, "Larq: An open-source library for training binarized neural networks," *Journal of Open Source Software*, vol. 5, no. 45, p. 1746, 2020. DOI: 10.21105/joss.01746.

[61] A. Pappalardo, *Xilinx/brevitas*, 2023. DOI: 10.5281/zenodo.3333552.

[62] PULP Platform, *PULP Platform*, available at https://www.pulp-platform.org/ (25/01/2024).

[63] F. Conti, D. Rossi, A. Pullini, I. Loi, and L. Benini, "PULP: A ultra-low power parallel accelerator for energy-efficient and flexible embedded vision," *J. Signal Process. Syst.*, vol. 84, no. 3, pp. 339–354, Sep. 2016. DOI: 10.1007/s11265-015-1070-9.

[64] F. Conti and A. Di Mauro, *NEMO (NEural Minimizer for pytOrch)*, available at https://github.com/pulp-platform/nemo (25/01/2024).

[65] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *2017 Conference on Neural Information Processing Systems (NIPS 2017)*, 2017.

[66] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2019. DOI: 10.5555/3454287.3455008.

[67] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, *PACT: Parameterized clipping activation for quantized neural networks*, 2018. DOI: 10.48550/arXiv.1805.06085.

[68] *torch.nn.Module*, available at https://pytorch.org/docs/stable/generated/torch.nn.Module.html (25/01/2024).

[69] *Automatic Differentiation Package - torch.autograd*, available at https://pytorch.org/docs/stable/autograd.html (25/01/2024).

[70] M. Spallanzani, G. Rutishauser, M. Scherer, P. Wiese, and F. Conti, *QuantLib*, available at https://github.com/pulp-platform/quantlib (25/01/2024).

[71] M. Spallanzani, G. Rutishauser, M. Scherer, P. Wiese, and F. Conti, *QuantLab*, available at https://github.com/pulp-platform/quantlab (25/01/2024).

[72] M. Spallanzani, G. Rutishauser, M. Scherer, A. Burrello, F. Conti, and L. Benini, *QuantLab: A modular framework for training and deploying mixed-precision NNs*, available at https://cms.tinyml.org/wp-content/uploads/talks2022/Spallanzani-Matteo-Hardware.pdf (25/01/2024).

[73] *Horovod on GPU*, available at https://horovod.readthedocs.io/en/stable/gpus_include.html (25/01/2024).

[74] M. Spallanzani, G. P. Leonardi, and L. Benini, "Training quantised neural networks with STE variants: The additive noise annealing algorithm," in *2022 IEEE / CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 470–479. DOI: 10.1109/CVPR52688.2022.00056.

[75] P. Busia, "Optimizing neural networks for embedded edge-processing platforms," Ph.D. dissertation, University of Cagliari, Cagliari, Italy, available at https://iris.unica.it/handle/11584/357302 (25/01/2024), 2023.

[76] T. M. Ingolfsson, U. Chakraborty, X. Wang, S. Beniczky, P. Ducouret, S. Benatti, P. Ryvlin, A. Cossettini, and L. Benini, "EpiDeNet: An energy-efficient approach to seizure detection for embedded systems," in *2023 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2023, pp. 1–5. DOI: 10.1109/BioCAS58349.2023.10388554.

[77] L. Zanatta, A. Di Mauro, F. Barchi, A. Bartolini, L. Benini, and A. Acquaviva, "Directly-trained spiking neural networks for deep reinforcement learning: Energy efficient implementation of event-based obstacle avoidance on a neuromorphic accelerator," *Neurocomputing*, vol. 562, p. 126 885, 2023. DOI: 10.1016/j.neucom.2023.126885.

[78] A. Burrello, F. Conti, L. Macan, G. Rutishauer, T. M. Ingolfsson, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, and L. Benini, *DORY: Deployment ORiented to memorY*, available at https://github.com/pulp-platform/dory (25/01/2024).

[79] N. Bruschi, G. Haugou, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, "GV-SoC: A highly configurable, fast and accurate full-platform simulator for RISC-V based IoT processors," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, 2021, pp. 409–416. DOI: 10.1109/ICCD53106.2021.00071.

[80] D. Farina and A. Holobar, "Characterization of human motor units from surface EMG decomposition," *Proceedings of the IEEE*, vol. 104, no. 2, pp. 353–373, 2016. DOI: 10.1109/JPROC.2015.2498665.

[81] M. J. Cheok, Z. B. Omar, and M. H. Jaward, "A review of hand gesture and sign language recognition techniques," *International Journal of Machine Learning and Cybernetics*, vol. 10, pp. 131–153, 2019. DOI: 10.1007/s13042-017-0705-5.

[82] R. Meattini, S. Benatti, U. Scarcia, D. De Gregorio, L. Benini, and C. Melchiorri, "An sEMG-based human–robot interface for robotic hands using machine learning and synergies," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 8, no. 7, pp. 1149–1158, 2018. DOI: 10.1109/TCPMT.2018.2799987.

[83] T. S. Saponas, D. S. Tan, D. Morris, J. Turner, and J. A. Landay, "Making muscle-computer interfaces more practical," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '10, Association for Computing Machinery, 2010, pp. 851–854. DOI: 10.1145/1753326.1753451.

[84] L. Guo, Z. Lu, and L. Yao, "Human-machine interaction sensing technology based on hand gesture recognition: A review," *IEEE Transactions on Human-Machine Systems*, vol. 51, no. 4, pp. 300–309, 2021. DOI: 10.1109/THMS.2021.3086003.

[85] T. Starner, J. Weaver, and A. Pentland, "Real-time american sign language recognition using desk and wearable computer based video," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 12, pp. 1371–1375, 1998. DOI: 10.1109/34.735811.

[86] T. S. Saponas, D. S. Tan, D. Morris, R. Balakrishnan, J. Turner, and J. A. Landay, "Enabling always-available input with muscle-computer interfaces," in *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '09, Association for Computing Machinery, 2009, pp. 167–176. DOI: 10.1145/1622176.1622208.

[87] Össur, *i-Limb Ultra*, available at https://www.ossur.com/en-us/prosthetics/arms/i-limb-ultra (24/01/2024).

[88] Ottobock, *Myoelectric prosthetics*, available at https://www.ottobockus.com/prosthetics/upper-limb-prosthetics/solution-overview/myoelectric-prosthetics/ (24/01/2024).

[89] J. Yousefi and A. Hamilton-Wright, "Characterizing EMG data using machine-learning tools," *Computers in Biology and Medicine*, vol. 51, pp. 1–13, 2014. DOI: 10.1016/j.compbiomed.2014.04.018.

[90] P. Kaufmann, K. Englehart, and M. Platzner, "Fluctuating EMG signals: Investigating long-term effects of pattern matching algorithms," in *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*, 2010, pp. 6357–6360. DOI: 10.1109/IEMBS.2010.5627288.

[91] M. Atzori, A. Gijsberts, C. Castellini, B. Caputo, A.-G. M. Hager, S. Elsig, G. Giatsidis, F. Bassetto, and H. Müller, "Electromyography data for non-invasive naturally-controlled robotic hand prostheses," *Scientific Data*, vol. 1, no. 1, Dec. 2014. DOI: 10.1038/sdata.2014.53.

[92] B. Milosevic, E. Farella, and S. Benatti, "Exploring arm posture and temporal variability in myoelectric hand gesture recognition," in *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, 2018, pp. 1032–1037. DOI: 10.1109/BIOROB.2018.8487838.

[93] L. G. Tassinary, J. T. Cacioppo, and E. J. Vanman, "The skeletomotor system: Surface electromyography," in *Handbook of Psychophysiology*, J. T. Cacioppo, L. G. Tassinary, and G. Berntson, Eds. Cambridge University Press, 2000, pp. 267–300. DOI: 10.1017/cbo9780511546396.012.

[94] C. J. D. Luca, "The use of surface electromyography in biomechanics," *Journal of Applied Biomechanics*, vol. 13, no. 2, pp. 135–163, 1997. DOI: 10.1123/jab.13.2.135.

[95] R. M. Rangayyan, *Introduction to Biomedical Signals*. 2002. DOI: 10.1109/9780470544204.ch1.

[96] M. Tomasini, S. Benatti, B. Milosevic, E. Farella, and L. Benini, "Power line interference removal for high-quality continuous biosignal monitoring with low-power wearable devices," *IEEE Sensors Journal*, vol. 16, no. 10, pp. 3887–3895, 2016. DOI: 10.1109/JSEN.2016.2536363.

[97] S. Benatti, F. Casamassima, B. Milosevic, E. Farella, P. Schönle, S. Fateh, T. Burger, Q. Huang, and L. Benini, "A versatile embedded platform for EMG acquisition and gesture recognition," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 9, no. 5, pp. 620–630, 2015. DOI: 10.1109/TBCAS.2015.2476555.

[98] Y. Hu, Y. Wong, W. Wei, Y. Du, M. Kankanhalli, and W. Geng, "A novel attention-based hybrid CNN-RNN architecture for sEMG-based gesture recognition," *PLOS ONE*, vol. 13, pp. 1–18, Oct. 2018. DOI: 10.1371/journal.pone.0206049.

[99] P. Tsinganos., B. Cornelis., J. Cornelis., B. Jansen., and A. Skodras., "Deep learning in EMG-based gesture recognition," in *Proceedings of the 5th International Conference on Physiological Computing Systems - PhyCS*, INSTICC, SciTePress, 2018, pp. 107–114. DOI: 10.5220/0006960201070114.

[100] L. Nieuwoudt and C. Fisher, "Investigation of real-time control of finger movements utilizing surface EMG signals," *IEEE Sensors Journal*, vol. 23, no. 18, pp. 21 989–21 997, 2023. DOI: 10.1109/JSEN.2023.3299384.

[101] L. Tong, M. Zhang, H. Ma, C. Wang, and L. Peng, "sEMG-based gesture recognition method for coal mine inspection manipulator using multistream CNN," *IEEE Sensors Journal*, vol. 23, no. 10, pp. 11 082–11 090, 2023. DOI: 10.1109/JSEN.2023.3264646.

[102] O. Kerdjidj, K. Amara, F. Harizi, and H. Boumridja, "Implementing hand gesture recognition using EMG on the Zynq circuit," *IEEE Sensors Journal*, vol. 23, no. 9, pp. 10 054–10 061, 2023. DOI: 10.1109/JSEN.2023.3259150.

[103] B. Hudgins, P. Parker, and R. Scott, "A new strategy for multifunction myoelectric control," *IEEE Transactions on Biomedical Engineering*, vol. 40, no. 1, pp. 82–94, 1993. DOI: 10.1109/10.204774.

[104] K. Englehart and B. Hudgins, "A robust, real-time control scheme for multifunction myoelectric control," *IEEE Transactions on Biomedical Engineering*, vol. 50, no. 7, pp. 848–854, Jul. 2003. DOI: 10.1109/TBME.2003.813539.

[105] C. Castellini, E. Gruppioni, A. Davalli, and G. Sandini, "Fine detection of grasp force and posture by amputees via surface electromyography," *Journal of Physiology - Paris*, vol. 103, no. 3, pp. 255–262, 2009. DOI: 10.1016/j.jphysparis.2009.08.008.

[106] A. Phinyomark and E. Scheme, "EMG pattern recognition in the era of big data and deep learning," *Big Data and Cognitive Computing*, vol. 2, no. 3, 2018. DOI: 10.3390/bdcc2030021.

[107] F. Palermo, M. Cognolato, A. Gijsberts, H. Müller, B. Caputo, and M. Atzori, "Repeatability of grasp recognition for robotic hand prosthesis control based on sEMG data," in *2017 International Conference on Rehabilitation Robotics (ICORR)*, 2017, pp. 1154–1159. DOI: 10.1109/ICORR.2017.8009405.

[108] S. Benatti, E. Farella, E. Gruppioni, and L. Benini, "Analysis of robust implementation of an EMG pattern recognition based control," in *Proceedings of the International Joint Conference on Biomedical Engineering Systems and Technologies - Volume 4*, ser. BIOSTEC 2014, SCITEPRESS - Science and Technology Publications, Lda, 2014, pp. 45–54. DOI: 10.5220/0004800300450054.

[109] V. H. Cene, M. Tosin, J. Machado, and A. Balbinot, "Open database for accurate upper-limb intent detection using electromyography and reliable extreme learning machines," *Sensors*, vol. 19, no. 8, 2019. DOI: 10.3390/s19081864.

[110] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[111] K.-H. Park and S.-W. Lee, "Movement intention decoding based on deep learning for multiuser myoelectric interfaces," pp. 1–2, 2016. DOI: 10.1109/IWW-BCI.2016.7457459.

[112] P. Tsinganos, B. Cornelis, J. Cornelis, B. Jansen, and A. Skodras, "Improved gesture recognition based on sEMG signals and TCN," pp. 1169–1173, 2019. DOI: 10.1109/ICASSP.2019.8683239.

[113] J. L. Betthauser, J. T. Krall, R. R. Kaliki, M. S. Fifer, and N. V. Thakor, "Stable electromyographic sequence prediction during movement transitions using temporal convolutional networks," pp. 1046–1049, 2019. DOI: 10.1109/NER.2019.8717169.

[114] Y. Du, W. Jin, W. Wei, Y. Hu, and W. Geng, "Surface EMG-based inter-session gesture recognition enhanced by deep domain adaptation," *Sensors*, vol. 17, no. 3, 2017. DOI: 10.3390/s17030458.

[115] M. Atzori, M. Cognolato, and H. Müller, "Deep learning with convolutional neural networks applied to electromyography data: A resource for the classification of movements for prosthetic hands," *Frontiers in Neurorobotics*, vol. 10, 2016. DOI: 10.3389/fnbot.2016.00009.

[116] Y. Li, N. Wang, J. Shi, J. Liu, and X. Hou, *Revisiting batch normalization for practical domain adaptation*, 2016. DOI: 10.48550/arXiv.1603.04779.

[117] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, JMLR.org, 2015, pp. 448–456. DOI: 10.5555/3045118.3045167.

[118] W. Wei, Y. Wong, Y. Du, Y. Hu, M. Kankanhalli, and W. Geng, "A multi-stream convolutional neural network for sEMG-based gesture recognition in muscle-computer interface," *Pattern Recognition Letters*, vol. 119, pp. 131–138, 2019. DOI: 10.1016/j.patrec.2017.12.005.

[119] B. Milosevic, S. Benatti, and E. Farella, "Design challenges for wearable EMG applications," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017, pp. 1432–1437. DOI: 10.23919/DATE.2017.7927217.

[120] A. Krasoulis, S. Vijayakumar, and K. Nazarpour, "Effect of user practice on prosthetic finger control with an intuitive myoelectric decoder," *Frontiers in Neuroscience*, vol. 13, 2019. DOI: 10.3389/fnins.2019.00891.

[121] P. Koch, M. Dreier, A. Larsen, T. J. Parbs, M. Maass, H. Phan, and A. Mertins, "Regression of hand movements from sEMG data with recurrent neural networks," in *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, 2020, pp. 3783–3787. DOI: 10.1109/EMBC44109.2020.9176278.

[122] A. Krasoulis and K. Nazarpour, "Myoelectric digit action decoding with multi-output, multi-class classification: An offline analysis," *Scientific Reports*, vol. 10, no. 1, Oct. 2020. DOI: 10.1038/s41598-020-72574-7.

[123] C. Castellini and P. van der Smagt, "Surface EMG in advanced hand prosthetics," *Biological Cybernetics*, vol. 100, no. 1, pp. 35–47, Nov. 2008. DOI: 10.1007/s00422-008-0278-1.

[124] International Electrotechnical Commission (IEC), *IEC 61508-1:2010, Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements*, available at https://webstore.iec.ch/publication/5515 (25/01/2024), Edition 2.0, 2010.

[125] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," *IEEE Access*, vol. 7, pp. 53 040–53 065, 2019. DOI: 10.1109/ACCESS.2019.2912200.

[126] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An overview on edge computing research," *IEEE Access*, vol. 8, pp. 85 714–85 728, 2020. DOI: 10.1109/ACCESS.2020.2991734.

[127] F. Wang, M. Zhang, X. Wang, X. Ma, and J. Liu, "Deep learning for edge computing applications: A state-of-the-art survey," *IEEE Access*, vol. 8, pp. 58 322–58 336, 2020. DOI: 10.1109/ACCESS.2020.2982411.

[128] L. Dutta and S. Bharali, "TinyML meets IoT: A comprehensive survey," *Internet of Things*, vol. 16, p. 100 461, 2021. DOI: 10.1016/j.iot.2021.100461.

[129] P. P. Ray, "A review on TinyML: State-of-the-art and prospects," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 4, pp. 1595–1623, 2022. DOI: 10.1016/j.jksuci.2021.11.019.

[130] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A comprehensive survey on TinyML," *IEEE Access*, pp. 1–1, 2023. DOI: 10.1109/ACCESS.2023.3294111.

[131] A. Burrello, M. Scherer, M. Zanghieri, F. Conti, and L. Benini, "A microcontroller is all you need: Enabling transformer execution on low-power IoT endnodes," in *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, 2021, pp. 1–6. DOI: 10.1109/COINS51742.2021.9524173.

[132] R. S. Peres, X. Jia, J. Lee, K. Sun, A. W. Colombo, and J. Barata, "Industrial artificial intelligence in Industry 4.0 - systematic review, challenges and outlook," *IEEE Access*, vol. 8, pp. 220 121–220 139, 2020. DOI: 10.1109/ACCESS.2020.3042874.

[133] *Artificial intelligence for digitizing industry*, May 2019. DOI: 10.3030/826060.

[134] International Electrotechnical Commission (IEC), *IEC 61496-1:2020, Safety of machinery electro-sensitive protective equipment - Part 1: General requirements and tests*, available at https://webstore.iec.ch/publication/63115 (25/01/2024), Edition 4.0, 2020.

[135] F. Conti, F. Indirli, A. Latella, F. Papariello, G. M. Puglia, F. Tecce, G. Urlini, and M. Zanghieri, "AI-powered collision avoidance safety system for industrial woodworking machinery," in *AI4DI – Applications*. River Publishers, 2021. DOI: 10.1201/9781003337232-17.

[136] SCM Group, *Morbidelli M100/M200. numerical-controlled machining centres*, available at https://www.scmgroup.com/en/scmwood/products/machining-centres.c874/cnc-machining-centres-for-routing-and-drilling.878/morbidelli-m100-200.32314 (25/01/2024), 2022.

[137] HOMAG Group, *CNC processing center CENTATEQ P-210*, available at https://www.homag.com/en/product-detail/cnc-processing-center-centateq-p-210 (25/01/2024), 2023.

[138] Biesse Group, *Rover K FT. CNC machining centre*, available at https://www.biesse.com/ww/wood/cnc-work-centres/rover-k-ft (25/01/2024), 2023.

[139] Biesse Group, *Rover K Smart. CNC machining centre*, available at https://www.biesse.com/ww/wood/cnc-work-centres/rover-k-smart (25/01/2024), 2023.

[140] Biesse Group, *Rover A 12/15/18. CNC machining centre*, available at https://www.biesse.com/ww/wood/cnc-work-centres/rover-a-1215 (25/01/2024), 2023.

[141] International Electrotechnical Commission (IEC), *IEC 62046:2018, Safety of machinery - Application of protective equipment to detect the presence of persons*, available at https://webstore.iec.ch/publication/27263 (25/01/2024), Edition 1.0, 2018.

[142] SCM Group, *Morbidelli X200/X400. CNC nesting machining centres for drilling and routing*, available at https://www.scmgroup.com/products/docs/CDL/Morbidelli_x200-x400_rev00_mag19_Ing.pdf (25/01/2024), 2019.

[143] Höchsmann, *SCM Morbidelli X200*, available at https://wtp.hoechsmann.com/it/lexikon/40505/morbidelli_x200 (25/01/2024).

[144] K. M. Fiest, K. M. Sauro, S. Wiebe, S. B. Patten, C.-S. Kwon, J. Dykeman, T. Pringsheim, D. L. Lorenzetti, and N. Jetté, "Prevalence and incidence of epilepsy: A systematic review and meta-analysis of international studies," *Neurology*, vol. 88, no. 3, pp. 296–303, 2017. DOI: 10.1212/WNL.0000000000003509.

[145] L. Kalilani, X. Sun, B. Pelgrims, M. Noack-Rink, and V. Villanueva, "The epidemiology of drug-resistant epilepsy: A systematic review and meta-analysis," *Epilepsia*, vol. 59, no. 12, pp. 2179–2193, DOI: 10.1111/epi.14596.

[146] M. Hirsch, D.-M. Altenmüller, and A. Schulze-Bonhage, "Latencies from intracranial seizure onset to ictal tachycardia: A comparison to surface EEG patterns and other clinical signs," *Epilepsia*, vol. 56, no. 10, pp. 1639–1647, DOI: 10.1111/epi.13117.

[147] C. Rummel, E. Abela, R. G. Andrzejak, M. Hauf, C. Pollo, M. Müller, C. Weisstanner, R. Wiest, and K. Schindler, "Resected brain tissue, seizure onset zone and quantitative EEG measures: Towards prediction of post-surgical seizure control," *PLOS ONE*, vol. 10, pp. 1–26, Oct. 2015. DOI: 10.1371/journal.pone.0141023.

[148] B. C. Munsell, C.-Y. Wee, S. S. Keller, B. Weber, C. Elger, L. A. T. da Silva, T. Nesland, M. Styner, D. Shen, and L. Bonilha, "Evaluation of machine learning algorithms for treatment outcome prediction in patients with epilepsy based on structural connectome data," *NeuroImage*, vol. 118, pp. 219–230, 2015. DOI: 10.1016/j.neuroimage.2015.06.008.

[149] A. K. Jaiswal and H. Banka, "Local pattern transformation based feature extraction techniques for classification of epileptic EEG signals," *Biomedical Signal Processing and Control*, vol. 34, pp. 81–92, 2017. DOI: 10.1016/j.bspc.2017.01.005.

[150] S. N. Baldassano, B. H. Brinkmann, H. Ung, T. Blevins, E. C. Conrad, K. Leyde, M. J. Cook, A. N. Khambhati, J. B. Wagenaar, G. A. Worrell, and B. Litt, "Crowdsourcing seizure detection: algorithm development and validation on human implanted device recordings," *Brain*, vol. 140, no. 6, pp. 1680–1691, Apr. 2017. DOI: 10.1093/brain/awx098.

[151] Y. Yuan, G. Xun, K. Jia, and A. Zhang, "A multi-view deep learning framework for EEG seizure detection," *IEEE Journal of Biomedical and Health Informatics*, vol. 23, no. 1, pp. 83–94, 2019. DOI: 10.1109/JBHI.2018.2871678.

[152] R. Hussein, H. Palangi, Z. J. Wang, and R. Ward, "Robust detection of epileptic seizures using deep neural networks," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 2546–2550. DOI: 10.1109/ICASSP.2018.8462029.

[153] Y. Nagahama, A. J. Schmitt, D. Nakagawa, A. S. Vesole, J. Kamm, C. K. Kovach, D. Hasan, M. Granner, B. J. Dlouhy, M. A. Howard, and H. Kawasaki, "Intracranial EEG for seizure focus localization: Evolving techniques, outcomes, complications, and utility of combining surface and depth electrodes," *Journal of Neurosurgery*, vol. 130, no. 4, pp. 1180–1192, 2019. DOI: 10.3171/2018.1.JNS171808.

[154] A. Burrello, S. Benatti, K. Schindler, L. Benini, and A. Rahimi, "An ensemble of hyperdimensional classifiers: Hardware-friendly short-latency seizure detection with automatic iEEG electrode selection," *IEEE Journal of Biomedical and Health Informatics*, vol. 25, no. 4, pp. 935–946, 2021. DOI: 10.1109/JBHI.2020.3022211.

[155] W. Stacey, M. Le Van Quyen, F. Mormann, and A. Schulze-Bonhage, "What is the present-day EEG evidence for a preictal state?" *Epilepsy Research*, vol. 97, no. 3, pp. 243–251, 2011, Special Issue on Epilepsy Research UK Workshop 2010 on "Preictal Phenomena". DOI: 10.1016/j.eplepsyres.2011.07.012.

[156] A. Bablani, D. R. Edla, D. Tripathi, and R. Cheruku, "Survey on brain-computer interface: An emerging computational intelligence paradigm," *ACM Comput. Surv.*, vol. 52, no. 1, Feb. 2019. DOI: 10.1145/3297713.

[157] J. Parvizi and S. Kastner, "Promises and limitations of human intracranial electroencephalography," *Nature neuroscience*, vol. 21, no. 4, pp. 474–483, 2018. DOI: 10.1038/s41593-018-0108-2.

[158] J. Becedas, "Brain–machine interfaces: Basis and advances," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 825–836, 2012. DOI: 10.1109/TSMCC.2012.2203301.

[159] W. A. Ríos-Herrera, P. V. Olguín-Rodríguez, J. D. Arzate-Mena, M. Corsi-Cabrera, J. Escalona, A. Marín-García, J. Ramos-Loyo, A. L. Rivera, D. Rivera-López, J. F. Zapata-Berruecos, and M. F. Müller, "The influence of EEG references on the analysis of spatio-temporal interrelation patterns," *Frontiers in Neuroscience*, vol. 13, 2019. DOI: 10.3389/fnins.2019.00941.

[160] V. J. Lawhern, A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, and B. J. Lance, "EEGNet: A compact convolutional neural network for EEG-based brain–computer interfaces," *Journal of Neural Engineering*, vol. 15, no. 5, p. 056 013, Jul. 2018. DOI: 10.1088/1741-2552/aace8c.

[161] N. Waytowich, V. J. Lawhern, J. O. Garcia, J. Cummings, J. Faller, P. Sajda, and J. M. Vettel, "Compact convolutional neural networks for classification of asynchronous steady-state visual evoked potentials," *Journal of Neural Engineering*, vol. 15, no. 6, p. 066 031, Oct. 2018. DOI: 10.1088/1741-2552/aae5d8.

[162] T. Schneider, X. Wang, M. Hersche, L. Cavigelli, and L. Benini, "Q-EEGNet: An energy-efficient 8-bit quantized parallel EEGNet implementation for edge motor-imagery brain-machine interfaces," in *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2020, pp. 284–289. DOI: 10.1109/SMARTCOMP50058.2020.00065.

[163] G. M. Schroeder, B. Diehl, F. A. Chowdhury, J. S. Duncan, J. de Tisi, A. J. Trevelyan, R. Forsyth, A. Jackson, P. N. Taylor, and Y. Wang, "Seizure pathways change on circadian and slower timescales in individual patients with focal epilepsy," *Proceedings of the National Academy of Sciences*, vol. 117, no. 20, pp. 11 048–11 058, 2020. DOI: 10.1073/pnas.1922084117.

[164] P. Q. Duy, G. L. Krauss, N. E. Crone, M. Ma, and E. L. Johnson, "Antiepileptic drug withdrawal and seizure severity in the epilepsy monitoring unit," *Epilepsy & Behavior*, vol. 109, p. 107 128, 2020. DOI: 10.1016/j.yebeh.2020.107128.

[165] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014. DOI: 10.5555/2627435.2670313.

[166] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Nov. 2011. DOI: 10.5555/1953048.2078195.

[167] M. Atzori, A. Gijsberts, S. Heynen, A.-G. M. Hager, O. Deriaz, P. van der Smagt, C. Castellini, B. Caputo, and H. Müller, "Building the ninapro database: A resource for the biorobotics community," in *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, 2012, pp. 1258–1265. DOI: 10.1109/BioRob.2012.6290287.

[168] S. Amsüss, L. P. Paredes, N. Rudigkeit, B. Graimann, M. J. Herrmann, and D. Farina, "Long term stability of surface EMG pattern classification for prosthetic control," in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, IEEE, 2013. DOI: 10.1109/embc.2013.6610327.

[169] J. He, D. Zhang, N. Jiang, X. Sheng, D. Farina, and X. Zhu, "User adaptation in long-term, open-loop myoelectric training: Implications for EMG pattern recognition in prosthesis control," *Journal of Neural Engineering*, vol. 12, no. 4, p. 046 005, Jun. 2015. DOI: 10.1088/1741-2560/12/4/046005.

[170] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "PULP-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, no. 2164, p. 20 190 155, Dec. 2019. DOI: 10.1098/rsta.2019.0155.

[171] J. Liu, F. Zhang, and H. H. Huang, "An open and configurable embedded system for EMG pattern recognition implementation for artificial arms," in *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2014, pp. 4095–4098. DOI: 10.1109/EMBC.2014.6944524.

[172] X. Zhang, H. Huang, and Q. Yang, "Real-time implementation of a self-recovery EMG pattern recognition interface for artificial arms," in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2013, pp. 5926–5929. DOI: 10.1109/EMBC.2013.6610901.

[173] Texas Instruments, *ADS129x low-power, 8-channel, 24-bit analog front-end for biopotential measurements*, available at http://www.ti.com/lit/ds/symlink/ads1298.pdf (24/01/2024), 2015.

[174] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "A 64-mW DNN-based visual navigation engine for autonomous nano-drones," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8357–8371, 2019. DOI: 10.1109/JIOT.2019.2917066.

[175] GreenWaves Technologies, *GAP SDK*, available at https://greenwaves-technologies.com/setting-up-sdk/ (24/01/2024).

[176] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, 2017. DOI: 10.48550/arXiv.1704.04861.

[177] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

[178] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856. DOI: 10.1109/CVPR.2018.00716.

[179] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2818–2826. DOI: 10.1109/CVPR.2016.308.

[180] M. Rusci, A. Capotondi, F. Conti, and L. Benini, "Work-in-progress: Quantized NNs as the definitive solution for inference on low-power ARM MCUs?" In *2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2018, pp. 1–2. DOI: 10.1109/CODESISSS.2018.8525915.

[181] F. Glaser, G. Haugou, D. Rossi, Q. Huang, and L. Benini, "Hardware-accelerated energy-efficient synchronization and communication for ultra-low-power tightly coupled clusters," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 552–557. DOI: 10.23919/DATE.2019.8715266.

[182] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, and L. Van Gool, "AI benchmark: Running deep neural networks on Android smartphones," in *Computer Vision – ECCV 2018 Workshops*, L. Leal-Taixé and S. Roth, Eds., Springer International Publishing, 2019, pp. 288–314. DOI: 10.1007/978-3-030-11021-5_19.

[183] A. Burrello, M. Zanghieri, C. Sarti, L. Ravaglia, S. Benatti, and L. Benini, "Tackling time-variability in sEMG-based gesture recognition with on-device incremental learning and temporal convolutional networks," in *2021 IEEE Sensors Applications Symposium (SAS)*, 2021, pp. 1–6. DOI: 10.1109/SAS51076.2021.9530007.

[184] L. Pellegrini, G. Graffieti, V. Lomonaco, and D. Maltoni, "Latent replay for real-time continual learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 10 203–10 209. DOI: 10.1109/IROS45743.2020.9341460.

[185] R. N. Khushaba, “Correlation analysis of electromyogram signals for multiuser myoelectric interfaces,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 22, no. 4, pp. 745–755, 2014. DOI: 10.1109/TNSRE.2014.2304470.

[186] Z. Fan, Z. Wang, G. Li, and R. Wang, “A canonical correlation analysis based EMG classification algorithm for eliminating electrode shift effect,” in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2016, pp. 867–870. DOI: 10.1109/EMBC.2016.7590838.

[187] E. Donati, S. Benatti, E. Ceolini, and G. Indiveri, “Long-term stable electromyography classification using canonical correlation analysis,” in *2023 11th International IEEE/EMBS Conference on Neural Engineering (NER)*, 2023, pp. 1–4. DOI: 10.1109/NER52421.2023.10123768.

[188] S. Lee and S. Nirjon, “Learning in the wild: When, how, and what to learn for on-device dataset adaptation,” in *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, ser. AIChallengeIoT ’20, Association for Computing Machinery, 2020, pp. 34–40. DOI: 10.1145/3417313.3429382.

[189] A. Hyvärinen, J. Karhunen, and E. Oja, “Principal component analysis and whitening,” in *Independent Component Analysis*. John Wiley & Sons, Ltd, 2001, ch. 6, pp. 125–144. DOI: 10.1002/0471221317.ch6.

[190] M. Zanghieri, *sEMG-based hand gesture recognition with deep learning*, M.Sc. thesis, University of Bologna, Bologna, Italy, 2019. DOI: 10.48550/arXiv.2306.10954.

[191] M. Orlandi, M. Zanghieri, V. J. Kartsch Morinigo, F. Conti, D. Schiavone, L. Benini, and S. Benatti, “sEMG neural spikes reconstruction for gesture recognition on a low-power multicore processor,” in *2022 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2022, pp. 704–708. DOI: 10.1109/BioCAS54905.2022.9948617.

[192] NinaPro Team, *NinaPro DB8*, available at https://ninapro.hevs.ch/instructions/DB8.html (24/01/2024).

[193] T. Bao, Y. Zhao, S. A. R. Zaidi, S. Xie, P. Yang, and Z. Zhang, “A deep Kalman filter network for hand kinematics estimation using sEMG,” *Pattern Recognition Letters*, vol. 143, pp. 88–94, 2021. DOI: 10.1016/j.patrec.2021.01.001.

[194] U. Côté-Allard, C. L. Fall, A. Drouin, A. Campeau-Lecours, C. Gosselin, K. Glette, F. Laviolette, and B. Gosselin, "Deep learning for electromyographic hand gesture signal classification using transfer learning," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 27, no. 4, pp. 760–771, 2019. DOI: 10.1109/TNSRE.2019.2896269.

[195] R. C. Sîmpetru, M. Osswald, D. I. Braun, D. S. Oliveira, A. L. Cakici, and A. Del Vecchio, "Accurate continuous prediction of 14 degrees of freedom of the hand from myoelectrical signals through convolutive deep learning," in *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, 2022, pp. 702–706. DOI: 10.1109/EMBC48229.2022.9870937.

[196] R. C. Sîmpetru, A. Arkudas, D. I. Braun, M. Osswald, D. S. de Oliveira, B. Eskofier, T. M. Kinfe, and A. Del Vecchio, "Sensing the full dynamics of the human hand with a neural interface and deep learning," Cold Spring Harbor Laboratory, 2022. DOI: 10.1101/2022.07.29.502064.

[197] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997. DOI: 10.1016/S0893-6080(97)00011-7.

[198] A. Di Mauro, A. S. Prasad, Z. Huang, M. Spallanzani, F. Conti, and L. Benini, "SNE: An energy-proportional digital accelerator for sparse event-based convolutions," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 825–830. DOI: 10.23919/DATE54114.2022.9774552.

[199] A. Di Mauro, M. Scherer, D. Rossi, and L. Benini, "Kraken: A direct event/frame-based multi-sensor fusion SoC for ultra-efficient visual processing in nano-UAVs," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, 2022, pp. 1–19. DOI: 10.1109/HCS55958.2022.9895621.

[200] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 106–122, 2018. DOI: 10.1109/TBCAS.2017.2759700.

[201] E. Donati, M. Payvand, N. Risi, R. Krause, and G. Indiveri, "Discrimination of EMG signals using a neuromorphic implementation of a spiking neural network," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 5, pp. 795–803, 2019. DOI: 10.1109/TBCAS.2019.2925454.

[202] Y. Ma, E. Donati, B. Chen, P. Ren, N. Zheng, and G. Indiveri, "Neuromorphic implementation of a recurrent neural network for EMG classification," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2020, pp. 69–73. DOI: 10.1109/AICAS48895.2020.9073810.

[203] Y. Ma, B. Chen, P. Ren, N. Zheng, G. Indiveri, and E. Donati, "EMG-based gestures classification using a mixed-signal neuromorphic processing system," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 4, pp. 578–587, 2020. DOI: 10.1109/JETCAS.2020.3037951.

[204] E. Donati, M. Payvand, N. Risi, R. Krause, K. Burelo, G. Indiveri, T. Dalgaty, and E. Vianello, "Processing EMG signals using reservoir computing on an event-based neuromorphic system," in *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2018, pp. 1–4. DOI: 10.1109/BIOCAS.2018.8584674.

[205] A. Jiménez-Fernández, E. Cerezuela-Escudero, L. Miró-Amarante, M. J. Domínguez-Morales, F. de Asís Gómez-Rodríguez, A. Linares-Barranco, and G. Jiménez-Moreno, "A binaural neuromorphic auditory sensor for FPGA: A spike signal processing approach," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 4, pp. 804–818, 2017. DOI: 10.1109/TNNLS.2016.2583223.

[206] S.-C. Liu, A. van Schaik, B. A. Minch, and T. Delbrück, "Event-based 64-channel binaural silicon cochlea with Q enhancement mechanisms," in *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 2027–2030. DOI: 10.1109/ISCAS.2010.5537164.

[207] M. Yang, C.-H. Chien, T. Delbruck, and S.-C. Liu, "A 0.5V 55μW 64×2 channel binaural silicon cochlea for event-driven stereo-audio sensing," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 11, pp. 2554–2569, 2016. DOI: 10.1109/JSSC.2016.2604285.

[208] I. M. Park, S. Seth, A. R. Paiva, L. Li, and J. C. Principe, "Kernel methods on spike train space for neuroscience: A tutorial," *IEEE Signal Processing Magazine*, vol. 30, no. 4, pp. 149–160, 2013. DOI: 10.1109/MSP.2013.2251072.

[209] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *eLife*, vol. 8, F. K. Skinner, R. L. Calabrese, F. K. Skinner, F. Zeldenrust, and R. C. Gerkin, Eds., e47314, Aug. 2019. DOI: 10.7554/eLife.47314.

[210] X. Jiang, X. Liu, J. Fan, X. Ye, C. Dai, E. A. Clancy, M. Akay, and W. Chen, "Open access dataset, toolbox and benchmark processing results of high-density surface electromyogram recordings," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 29, pp. 1035–1046, 2021. DOI: 10.1109/TNSRE.2021.3082551.

[211] X. Jiang, X. Liu, J. Fan, C. Dai, E. A. Clancy, and W. Chen, "Random channel masks for regularization of least squares-based finger EMG-force modeling to improve cross-day performance," *IEEE Transactions on Neural Systems and*

Rehabilitation Engineering, vol. 30, pp. 2157–2167, 2022. DOI: 10.1109/TNSRE.2022.3194246.

[212] X. Jiang, K. Nazarpour, and C. Dai, “Explainable and robust deep forests for EMG-force modeling,” *IEEE Journal of Biomedical and Health Informatics*, vol. 27, no. 6, pp. 2841–2852, 2023. DOI: 10.1109/JBHI.2023.3262316.

[213] W. Wu, L. Jiang, B. Yang, K. Gong, C. Peng, and T. He, “A new EMG decomposition framework for upper limb prosthetic systems,” *Journal of Bionic Engineering*, Jul. 2023. DOI: 10.1007/s42235-023-00407-0.