CHALMERS
UNIVERSITY OF TECHNOLOGY

Deployment of Machine Learning Algorithms on Resource-Constrained Hardware Platforms for Prosthetics

Downloaded from: https://research.chalmers.se, 2024-10-12 06:08 UTC

Citation for the original published paper (version of record):
Just, F., Ghinami, C., Zbinden, J. et al (2024). Deployment of Machine Learning Algorithms on Resource-Constrained Hardware Platforms for Prosthetics. IEEE Access, 12: 40439-40449. http://dx.doi.org/10.1109/ACCESS.2024.3371251

N.B. When citing this work, cite the original published paper.

© 2024 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, or reuse of any copyrighted component of this work in other works.

(article starts on next page)

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 4 December 2023, accepted 15 January 2024, date of publication 27 February 2024, date of current version 21 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3371251

RESEARCH ARTICLE

Deployment of Machine Learning Algorithms on Resource-Constrained Hardware Platforms for Prosthetics

FABIAN JUST[1,2], (Member, IEEE), CHIARA GHINAMI[1], JAN ZBINDEN[1,2], (Graduate Student Member, IEEE), AND MAX ORTIZ-CATALAN[1,3,4,5], (Senior Member, IEEE)

[1]Center for Bionics and Pain Research, 431 30 Mölndal, Sweden
[2]Department of Electrical Engineering, Chalmers University of Technology, 412 96 Gothenburg, Sweden
[3]Bionics Institute, Melbourne, VIC 3002, Australia
[4]Medical Bionics Department, The University of Melbourne, Melbourne, VIC 3010, Australia
[5]Prometei Pain Rehabilitation Center, 21018 Vinnytsia, Ukraine

Corresponding author: Max Ortiz-Catalan (maxortizc@outlook.com)

The work of Fabian Just, Jan Zbinden, and Max Ortiz-Catalan was supported in part by the Promobilia Foundation; and in part by the IngaBritt and Arne Lundbergs Foundation.

ABSTRACT Motion intent recognition for controlling prosthetic systems has long relied on machine learning algorithms. Artificial neural networks have shown great promise for solving such nonlinear classification tasks, making them a viable method for this purpose. To bring these advanced methods and algorithms beyond the confines of the laboratory and into the daily lives of prosthetic users, self-contained embedded systems are essential. However, embedded systems face constraints in size, computational power, memory footprint, and power consumption, as they must be non-intrusive and discreetly integrated into commercial prosthetic components. One promising approach to tackle these challenges is to use network quantization, which allows complying with limitations without significant loss in accuracy. Here, we compare network quantization performance for self-contained systems using TensorFlow Lite and the recently developed QKeras platform. Due to internal libraries, the use of TensorFlow Lite led to a 8 times higher flash memory usage than that of the unquantized reference network, disadvantageous for self-contained prosthetic systems. In response, we offer open-source code solutions that leverage the QKeras platform, effectively reducing flash memory requirements by 24 times compared to Tensorflow Lite. Additionally, we conducted a comprehensive comparison of state-of-the-art microcontrollers. Our results reveal that the adoption of new architectures offers substantial reductions in inference time and power consumption. These improvements pave the way for real-time decoding of motor intent using more advanced machine learning algorithms for daily life usage, possibly enabling more reliable and precise control for prosthetic users.

INDEX TERMS Motion intent recognition, machine learning, neural networks, quantization, prosthetics, classification, embedded systems, real-time, QKeras, TensorFlow.

I. INTRODUCTION

Laboratory experiments have shown promising strategies to decode human motor intent to control bionic limbs [1], [2], [3], [4], [5], [6]. Mostly, electromyography (EMG) recordings are used to relate the activity of muscles remnant from amputation to a desired prosthesis joint actuation. Recent improvements in computational hardware, especially the computational power-to-size ratio, allowed for the acquisition and processing of EMG signals to be made more portable and reliable. This led several prosthetic control studies to be carried out outside lab settings, as well as many commercial applications like commercial prosthetics using advanced myoelectric algorithms [7]. For computational hardware in

The associate editor coordinating the review of this manuscript and approving it for publication was Nuno M. Garcia.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

prosthetics, microcontrollers (MCU) are commonly favored over other embedded systems like Field Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs) as the go-to platform because they are less complex to program and often feature extensive library and tool support [8], [9], [10]. Previously, standard machine learning algorithms like support vector machine (SVM) and linear discriminant analysis (LDA) were used on the computational hardware to decode motion intent [1]. Due to the availability of bigger motion intent data sets and the need for higher performance, the field shifted towards neural networks (NN), especially deep neural networks [2]. Deep learning techniques like deep neural networks enable more complex and more accurate motion intent recognition [11].

However, the current state-of-the-art neural networks imposes significant demands on memory, computation, and energy [12], [13]. This presents a challenge for portable, self-contained systems that operate all day with limited resources, relying solely on low-power MCUs. To address this issue and accommodate the limited computing capability and storage capacity of MCU-class devices, recent advancements in deep learning training methodologies have introduced novel quantization methods. These methods aim to compress the network's weights or activations into 8-bit or smaller data types while incurring minimal or even negligible accuracy loss [14], [15], [16], [17]. By employing these quantization techniques, known as network quantization, the memory requirements of the models are significantly reduced compared to their full precision counterparts.

Consequently, both industry and academia have been actively engaged in developing hardware and software platforms dedicated to the efficient execution of quantized neural networks (QNNs) on MCU-class devices. This concerted effort is driven by the recognition that QNNs offer a viable solution to enable the deployment of neural networks on resource-constrained systems, where memory, computation, and energy efficiency are paramount considerations. One of the most used platform for training and quantization is TensorFlow Lite [18].

This paper compares the advantages of quantization faced by self-contained systems when utilizing the TensorFlow Lite and Qkeras platform. The latter was made possible thanks to the proposed innovative deployment workflow targeting the newly released QKeras platform. We present a comprehensive QNN comparison tailored to Cortex-M-based microcontrollers and benchmark microcontrollers featuring DSP functionalities. The aim is to enhance the performance of self-contained embedded systems and simultaneously reducing flash memory footprint. As the results, we present the following key contributions in this paper:

1. We compared the quantization and deployment strategie for Cortex-M-based microcontrollers in TensorFlow Lite and QKeras. For the hardware targets, we measured inference time, energy consumption, and memory footprint.
2. We benchmarked different MCUs and compared the inference time of different data types
3. We show the potential of the newly released ARM Cortex-M55 processor with and without ARM Helium Vector Extension (VE)[1] for the computing load, in terms of inference time and memory footprint.
4. We provide open-source scripts[2] to facilitate the integration of our findings and to increase the usability of the novel QKeras platform with the CMSIS library (Common Microcontroller Software Interface Standard) [19].

II. RELATED WORK

Various research efforts have targeted machine learning and deep learning on resource-constrained devices and are presented in this section. Existing research can be grouped into embedded hardware for motion intent and network quantization methods.

A. EMBEDDED HARDWARE FOR MOTION INTENT RECOGNITION (DEEP LEARNING)

While the field of deep learning has been growing in terms of performance, network size, and training run time, the development of embedded hardware to process deep learning algorithms is struggling to keep up [20], [21]. As the demand for higher performance grows, researchers are exploring and investigating various options for implementation, including field-programmable gate array (FPGA) devices on systems-on-chip (SoCs) [22], [23], [24], [25], graphics processing units (GPUs) [26], [27], [28], and neuromorphic chips and processors [29], [30], [31].

However, when considering prosthetic devices that operate on batteries, the power envelope becomes a crucial constraint. FPGA SoCs and GPUs often exceed this power limitation, making them less feasible for such applications. On the one hand, neuromorphic technology offers the promise of exceptionally low power consumption, but several challenges still impede its rapid growth and widespread usage [32]. On the other hand, resource-constrained microcontrollers (MCUs) offer software programmability, affordability, and low power consumption. These unique attributes make MCUs well-suited for the development of portable prosthetic systems that are designed for all-day use [33]. Table 1 showcases numerous scientific papers that have utilized MCUs for EMG motion intent recognition.

The Cortex-M4 is the currently most used CPU in the field (see Table 1). In [34], a Raspberry Pi board is used to process the radial basis function (RBF) neural network classification algorithm. Both [35] and [38] have the highest time values, since they only reported their completion time including the finished movement of the prosthetic system (see Table 1). Our research gorup uses the Artificial Limb Controller (ALC) embedded system [36]. The system draws different currents depending on the operation mode, from idling (275 mW) to streaming data (495 mW) [36].

[1] https://www.arm.com/technologies/helium
[2] https://github.com/fabianjust/QNN-MCU

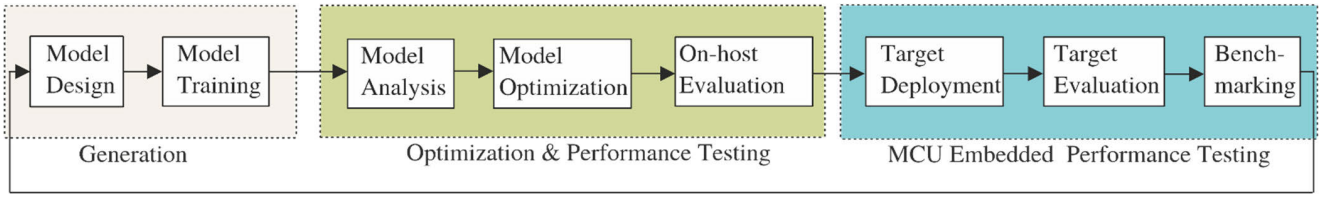

FIGURE 1. Design loop for development, optimization, deployment, and evaluation of neural networks for embedded microcontroller (MCU) use.

TABLE 1. MCU performance: Technical overview of EMG motion intent recognition systems. CT (completion time) includes the movement time of the prosthetic system.

Algorithm	CPU	Classification time [ms]	EMG channels	Power consumption [mW]
RBF[34]	4x Cortex-A53	4.5	8	-
LDA[35]	Cortex-M4	75(CT)	8	87
SVM[36]	Cortex-M4	3	8	275-495
ANN[37]	Cortex-M4	0.2	4	75
LDA[38]	Cortex-M4	200(CT)	8	-
LDA[39]	Cortex-A8	0.85	4	-

B. NETWORK QUANTIZATION

With the success of deep neural networks and their ever-increasing sizes, the quantization of neural networks has emerged as a fundamental technique to reduce model size and memory footprints. Remarkable progress in this area has led to quantized neural networks achieving similar levels of accuracy as their full-precision counterparts [14], [15], [17]. In neural network quantization, three key components can be targeted: weights, activations, and gradients [40]. In this study, our focus is on quantizing weights and activations.

While various approaches have explored 4-bit [41], [42], binary [43], [44], [45], [46], [47], and adaptive [48], [49] quantization, our work centers on 8-bit quantization, which is widely supported by most microcontrollers (MCUs). The quantization of parameters offers the following advantages:

1. *Smaller Model footprint*: With 8-bit quantization, we can reduce the model size by a factor of 4 (assuming an initial 32-bit model representation), with negligible accuracy loss.
2. *Less working memory and cache for activations*: Intermediate computations are typically stored in a cache for reuse by later layers of a deep network. Reducing the precision at which this data is stored leads to less working memory needed [41].
3. *Faster computation*: Most processors allow for faster processing of 8-bit data compared to 32-bit data.

The parameters quantization can either be performed by retraining the neural network model, a process that is called Quantization-Aware Training [50], [51], or done without re-training, a process that is often referred to as Post-Training Quantization [42], [52], [53]. In this work, we use the Quantization-Aware Training [40], [54] since it has proven to achieve higher accuracy value [41].

III. METHOD

A. FRAMEWORKS

We tested and compared the newly released QKeras framework [55] with the widely used TensorFlow Lite.

TensorFlow Lite is an open-source framework to enable machine learning models on embedded systems. TensorFlow Lite was designed to provide a unified ML framework, addressing the multitude of embedded platforms and hardware support [10]. Therefore, the TensorFlow library can run with or without CMSIS support, so it is hardware independent.

QKeras is a quantization extension to Keras. It allows the developer to choose the quantizer and the quantizer's parameters for each network layer leading to a customized deep quantized version of the Keras network [55], [56]. In our study, the QKeras implementation always uses the CMSIS support due to our developed support scripts.

B. TOOLCHAIN

As depicted in Fig.1, the initial phase involves the model design by selecting the network type and structure for testing purposes.

Subsequently, the model undergoes training and analysis (referred to as model training and analysis) to assess the baseline performance metrics, namely accuracy and byte size. At this stage, it is possible to examine the complexity of the model by considering the fixed number of parameters and estimated operations. The subsequent step entails optimizing the model to achieve reduced memory usage and reduced inference time (known as model optimization). By this point in the development, the model has been quantized. It is important to consider the hardware architecture of the target device during the quantization process. For instance, current MCU architectures cannot effectively utilize sub-int8 quantization. It should be noted that optimizations made during this phase permanently modify the neural network and have the potential to impact its accuracy. Hence, by conducting on-host evaluation, we can promptly evaluate the performance of the quantized model. Once the model is optimized, it's ready to be deployed on the MCU. The implementation on the MCU makes use of the CMSIS-NN library. This library, comprising optimized kernels, is an open-source solution designed to optimize the performance and minimize

TABLE 2. 3 layer (1 hidden layer) neural network architecture used for this study.

Layer (type)	Output shape	Parameters [#]
Dense 1	(None, 128)	1152
Dense 2	(None, 64)	8256
Dense 3	(None, 8)	520

the memory usage of neural network applications on Arm Cortex-M processors [13]. These optimized kernels do not change the numerical representation itself but increase the execution efficiency by leveraging target-specific features like single instruction, multiple data units (SMID).

The concluding steps involve testing the deployed model and evaluating its performance. Although the NN deployment does not modify the NN itself, the accuracy on the MCU can be altered by different underlying computations, potentially leading to varied classifications [13]. For this reason, it must be verified that the inference on the target hardware is identical to the inference on the host system (*target evaluation, see Fig. 1*). Furthermore, the inference time and the energy consumption are measured as part of the evaluation process. The tests were conducted on two distinct MCUs, employing different data types, and comparing the two implementations of the neural network, one utilizing TFLite on top of CMSIS-NN and the other without it (QKeras). Subsequently, the iterative development cycle recommences.

C. NETWORK AND EMG DATASET

A representative EMG dataset for motion intent with hand gestures in float32 data format was used for this study [57]. To classify motion intent for prosthetics, a simple feed-forward neural network (FFNN) with one hidden layer was chosen. This choice is substantiated by several surveys [58], [59] that highlight the widespread use of this model for motion intent recognition. The dataset used is balanced, this justifies the choice of the accuracy as a metric for the NN evaluation. In addition, the accuracy results as the most popular metric according to recent surveys [58], [59]. The network architecture is described in Table 2.

D. EXPERIMENT DESIGN

The experimental design workflow is depicted in Fig. 2. The comparison involves quantizing the float32 neural network to integer (int) types using one-time TensorFlow Lite and one-time QKeras.

As shown in Fig. 2, both experiment workflows initially utilize the Keras and QKeras tool for Quantization-Aware Training. In the first experiment, the trained Keras model is then converted into a TFLite model. This conversion process involves quantizing the model with int8 weights and uint8 activations, while keeping biases as int32 data type (mixed quantization). In TFLite, the neural network topology is deployed as microcode, which is interpreted at runtime rather than statically compiled. This process hinders compiler

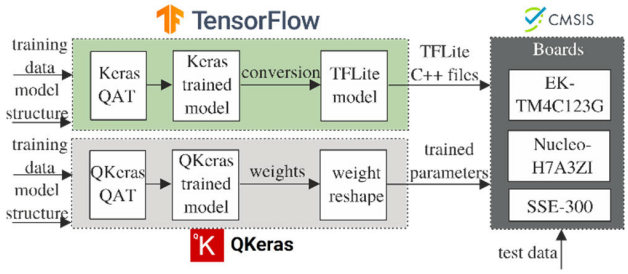

FIGURE 2. Experiment design using the Tensorflow Lite and QKeras platform for quantization of neural networks. Evaluation of the quantization performance is done on three MCU boards representing the neural interface market with usage of the CMSIS library.

optimizations and leads to increased memory usage [60]. TFLite can run with or without usage of the CMSIS-NN library.

For the QKeras experiment workflow, we conducted tests using QKeras for two homogeneous quantization methods: int8 (q7) and int16 (q15). Following Quantization-Aware Training, we manipulate the extracted parameters before being used as input for the CMSIS-NN functions. The manipulation is needed since the tensor layout differs between the machine learning framework and the CMSIS-NN library. The network parameters that the QKeras quantization produces are not integers, which is the format used by CMSIS. Therefore, bias and weight need to be converted to integers. Additionally, the layout of tensors follows a different convention in the QKeras compared to CMSIS-NN, so a reshape of the NN parameters was necessary before using them.

Importantly, only the parameter values and the CMSIS library significantly contribute to the flash memory footprint on the target hardware using QKeras compared to TFLite, which by itself occupies a significant part of the flash memory.

We tested the same implementation on three different hardware targets:

1. The TM4C123GHPM, from Texas Instrument, based on the Arm Cortex-M4 32-bit RISC core operating at up to 80 MHz used in the ALC developed by our group.
2. The STM32H7A3ZI from STMicroelectronics, based on the high-performance Arm Cortex-M7 32-bit RISC core operating at up to 280 MHz.
3. Corstone-300 (SSE-300) leverages Cortex-M55, Arm's most AI-capable Cortex-M processor and the first to feature Arm Helium vector processing technology. We tested the performance using a virtual hardware target since it is not released yet.

E. EXPERIMENT HARDWARE SETUP

Hardware tests were performed using the Keil MDK tool, which comprises an Integrated Development Environment called μ Vision specifically designed for Cortex-M devices, along with the ARM compiler. The tests were conducted with an operating frequency of 80 MHz. For each measurement, both the model and parameters were stored in the flash

memory, and the armclang compiler was used. Due to limited flash memory on TM4 using TensorFlow the Oz compiler optimization had to be used. All other measurements were carried out with the O2 compiler optimization.

Energy measurements were conducted using the ARM ULINKplus debugger. This debugger serves as a debug adapter that connects the PC's USB port to the target system through JTAG. It facilitates debugging, tracing, analysis, and the measurement of program consumption on the target hardware. Energy measurements were not done with the SSE-300, since it is only simulated and not released as hardware.

F. PRIMARY OUTCOME MEASURES

The three primary outcome measures of the study are inference time, flash memory footprint, and energy per inference, which are measured using Keil and Python.

Inference time: Inference time is defined as the duration required to execute a single motion intent classification on the trained neural network.

Flash memory footprint: The flash memory footprint is determined by the total memory required for the neural network code, library size, and initial parameters necessary for executing the motion intent classification.

Energy per inference: The energy per inference is calculated by multiplying the current consumed during classification by the 3.3V supply voltage over the inference time. Therefore,

$$\textit{Energy per inference} = \int_{0}^{\textit{inference time}} 3.3V \cdot \textit{current}\, dt$$

G. SECONDARY OUTCOME MEASURES

Accuracy loss and MCU accuracy loss: The MCU accuracy loss value is calculated by comparing the accuracy of the on-host QNN to the MCU accuracy of the deployed QNN.

IV. RESULTS

A. NETWORK SIZE REDUCTION

With TFLite, 65% on-host network size reduction is obtained (see Table 3). With QKeras, a network size reduction of 50% and 73% is achieved through int16 and int8 quantization strategies, respectively. The QNN, when deployed using CMSIS-NN, displays a decreased accuracy of less than 1% for all implementations ("Acc. loss" column in Table 3). The accuracy drop after the deployment in QKeras is slightly higher compared to when TFLite is used.

B. INFERENCE TIME

Using the CMSIS library in TensorFlow Lite results in a threefold inference time improvement compared to the hardware-independent solution with absence of CMSIS (see Table 4 and Fig. 3). Furthermore, when comparing all these results to our start scenario without quantization (none-float32), all configurations utilizing the CMSIS library demonstrated a minimum doubling of the speed and, in some cases, achieved up to three times faster inference time.

The new and, therefore, simulated Cortex-M55 processor (SSE-300) exhibits an inference time improvement of approximately 2 to 3 times when operated with helium. In general, SSE-300 with helium is double as fast as the STM32 and 4 times faster than the TM4.

TABLE 3. Memory footprint and accuracy loss of the network due to quantization with TensorFlow Lite (mixed quantization) or QKeras (q7,q15) Number of multiply-accumulates MACs = 9728 and parameters=9928.

Platform	None float32	TFLite	QKeras	QKeras
Quantization	-	mixed q	q15	q7
Acc. Loss [%]	-	0.3	0.36	0.88
MCU Loss [%]	-	0.1	0.7	0.5
Network size [kB]	37.8	13.14	19	9.9
Network size reduction [%]	-	65.2	49.7	73.8

TABLE 4. Inference time [μs] of the neural network on TM4, STM32, and SSE-300 Chips with various quantizations and CMSIS library usage. The O2 optimization was employed to accommodate limited flash memory of the MCU. SSE-300 was tested with and without Helium usage.

	None float32	TFLite	TFLite	QKeras	QKeras
Quantization	-	mixed q	mixed q	q15	q7
CMSIS	Yes	No	Yes	No	Yes
TM4	48.1	244	224	26	16.1
STM32	49.1	411 (O2)	383 (O2)	26.9	17
SSE-300 Helium	-	374 (O2, Hel)		24.3 (Hel)	14.2 (Hel)
SSE-300 No helium		420 (O2)		25.2	15.5

C. MEMORY FOOTPRINT

TFLite exhibits a substantial increase in flash memory consumption independent of the usage of the CMSIS library, as shown in Table 5 and Fig. 4. TFLite requires up to 24 times more flash memory compared to the QKeras implementation. The SSE-300 shows a 3% to 16% lower flash memory footprint than the TM4 or STM32 chip and the use of helium reduces the SSE-300 footprint by 4% up to 9%.

D. POWER CONSUMPTION

The TM4 chip exhibits a current consumption that is up to 106% higher than the STM32 chip, as depicted in Table 6. The q15 quantization of the neural network on both chips consumes the highest amount of current. In contrast, the unquantized float32 implementation consumes approximately 3% to 6% less current than the q15 implementation. The q7 implementation demonstrates the lowest power

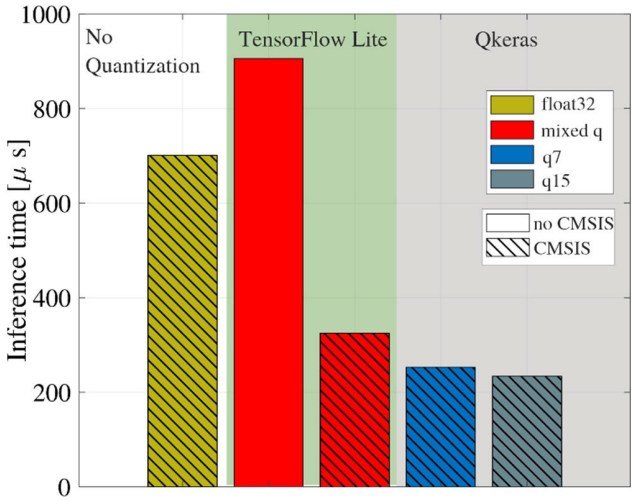

FIGURE 3. Inference time for the quantized networks in QKeras or TFlite on a STM32 chip with and without CMSIS library.

TABLE 5. Flash memory footprint [kB] of the neural network on TM4, STM32, and SSE-300 chips with various quantizations and CMSIS library usage. O2 optimization employed to accommodate limited flash memory of the MCU.

Platform	None float32	TFLite	TFLite	QKeras	QKeras
Quantization	-	mixed q	mixed q	q15	q7
CMSIS-NN	No	No	Yes	Yes	Yes
TM4	1165	574 (Oz)	1993 (Oz)	459	462
STM32	701	306	906	234	253
SSE-300 Helium	-	134 (Hel)	-	132 (Hel)	96 (Hel)
SSE-300		403		231	261

consumption, consuming 14% to 17% less current compared to q15.

The energy consumption per inference can be reduced by implementing quantization on the TM4 chip, achieving a reduction of 60%, and on the STM32 chip, achieving a reduction of 64% (see Fig. 5). Moreover, upgrading the chip architecture without applying quantization techniques leads to a noticeable reduction in the energy per inference, ranging from 71% to 73% (see Fig. 5). Notably, the combined utilization of quantization and chip upgrades in this case can yield an energy reduction of up to 90% per inference.

V. DISCUSSION

A. NETWORK SIZE REDUCTION

The quantization methods demonstrate an accuracy loss and MCU accuracy loss of less than 1% while achieving a significant network size reduction of 50-74% for each approach. This highlights the advantage of quantization in embedded neural networks on resource-constrained systems, where notable memory savings can be achieved without sacrificing performance. Specifically, the observed accuracy drop is more pronounced in QKeras compared to TFLite due to the manipulation and conversion of parameters to integers [13], [61]. A shown reduction of the deployed accuracy is not a feature of CMSIS-NN, but a random effect due to specific kernel implementations and a consequence of the different underlying computations [13], [61].

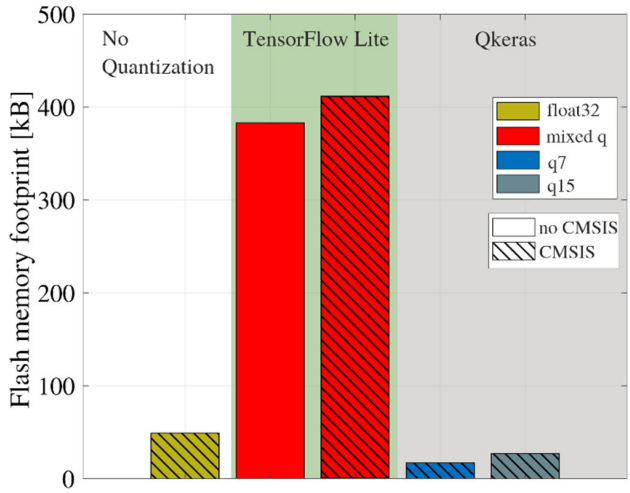

FIGURE 4. Flash memory footprint [kB] for the quantized neural networks with QKeras and TFLite on a STM32 chip with and without usage of the CMSIS library (only TFLite).

TABLE 6. Current consumption running the neural network on the TM4 and STM32 chips [mA].

Platform	None-f32	TFLite mixed q	QKeras q7	QKeras q15
CMSIS	Yes	Yes	Yes	Yes
TM4	20.5	-	18.1	21.1
STM32	9.8	8.7	8.68	10.4

B. INFERENCE TIME

The considerable reductions in inference time, by a factor of 2 to 3, directly result from the quantization of the neural network. TFLite leverages the CMSIS-NN library and exploits the hardware optimizations provided by CMSIS functions. Alternatively, it can operate independently of specific hardware but with a higher inference time, even surpassing that of the unquantized network. In the Cortex-M4 (TM4) and Cortex-M7-based microcontroller (STM32), the q15 inference time is up to 7% faster than the q7 inference time. This is due to the limitation of the microcontroller, as it can only perform dual 16-bit multiply-accumulates and cannot parallelize quad 8-bit signed multiply-accumulates. Consequently, the q7 inference implementation requires the expansion of q7 vectors into q15 vectors, making it slower than the q15 implementation.

C. MEMORY FOOTPRINT

The significant memory footprint of TensorFlow, which is 7 times higher than the unquantized network condition float32, predominantly stems from the TensorFlow library itself, as evidenced by the neural network model occupying only 13 KB (see Fig. 3). Flash memory consumption can be reduced by removing unused operations from the library

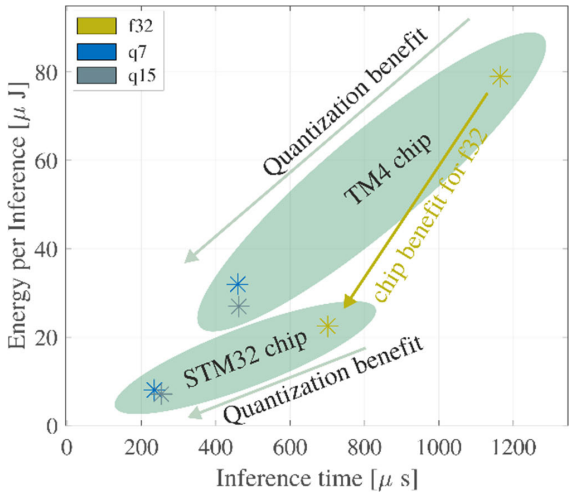

FIGURE 5. Quantization and chip benefit effects on the primary outcome measure energy per inference of float32 to q7 and q15 for the TM4 and STM32 chip.

and incorporating only the operations required by the current model. Notably, the utilization of QKeras together with our support for the CMSIS library results in a remarkable reduction in memory footprint. Thus, when memory footprint is a critical factor for the system, QKeras emerges as the preferred software solution. For the SSE-300 chip with and without helium, low-overhead branch instructions are available leading to much lower memory consumption than the STM32 and TM4 MCUs (see Table 5), improving even more the reduction of flash memory for resource-constrained embedded systems [62].

D. POWER CONSUMPTION

The q15 quantization leverages the SIMD architecture, SIMD instructions are likely to consume more power compared to the f32 scalar code. Since the 16-bit operations are more efficient, we can see these advantages in Fig.5. With the int8 data type, operations are still performed with 32-bit registers, but half of the register is unused since int8 data are half the int16 size. In this case, the switching activity of the transistors involved in the operations is lower, which leads to lower power consumption.

In the context of prosthetics, minimizing power consumption in self-contained prostheses is crucial to ensure functionality and independence in everyday life. This significance arises from the fact that active prosthetic users typically engage with their prostheses for an average of 11 hours daily, with merely 7 minutes dedicated to actual hand movement [63]. Even with periods of heightened energy consumption during active usage, the prosthetic system idles for a staggering 99% of the usage time, revealing an immense potential for power consumption savings. Therefore, by reducing the general power consumption through chip upgrades and network quantization (Fig. 5), substantial improvements in battery life can be achieved. Even more power consumption reductions can be achieved by incorporating the sleep mode of chips in the code.

TABLE 7. 3 layer (1 hidden layer) neural network architecture used for this study.

Layer (type)	Output shape	Parameters [#]
Dense 1	(None, 128)	1152
Dense 2	(None, 64)	8256
Dense 3	(None, 8)	520

E. GENERAL DISCUSSION

Quantization of neural networks plays a vital role in enabling resource-constrained embedded systems to effectively deploy larger networks with increased depth, thereby maximizing real-time classification accuracy. In scenarios where flash memory is limited, the QKeras platform emerges as the optimal software choice for network quantization. Therefore, detailed guidelines and scripts are provided in the supplementary material to facilitate the utilization of QKeras for feedforward neural networks.

The newly released Cortex-M55 processor proved to have the best flash memory footprint values in the simulation, due to efficiently exploiting the capabilities of int8 data type, and effectively handling ML workloads, due to the Helium vector extension. However, M55's performance should be also tested on board in hardware as soon it is available for purchase to have a correct comparison to the other MCU performances.

While this study focuses primarily on the quantization method for network compression, it is important to note that other compression techniques, such as pruning, are commonly employed. Future research should explore the effectiveness of these compression methods, which have shown promise and are now supported by both TensorFlow and QKeras frameworks.

VI. CONCLUSION

Quantization of neural networks with TensorFlow Lite leads to a massive increase in flash memory consumption. This poses a notable limitation for resource-constrained platforms, such as prosthetics designed for all-day wearability. QKeras with our CMSIS-NN support stands out by exclusively utilizing CMSIS-NN functions for quantization, therefore, delivering superior performance as TensorFlow Lite usable for resource-constrained platforms.

For the prosthetics community, we have provided open-source scripts and explanations to support the integration of QKeras and facilitate further development.[3]

APPENDIX A

Our integration begins with the definition and training of a neural network via QKeras.

The first step entails establishing the network's architecture and configuring the desired quantization. Representatively as in the paper, we use a three-layer feed-forward network as an example (see Table 7).

Notably, QKeras inherently facilitates network quantization, and we elaborate on the process of reducing the

[3]https://github.com/fabianjust/QNN-MCU

```
def create_qmodel(n_bits):

    model = Sequential()
    model.add(Input(shape=(8,)))

    model.add(QDense(128, name="fc1",
          kernel_quantizer=quantized_bits(n_bits, integer=1,     alpha="auto_po2"),
          bias_quantizer=quantized_bits(n_bits, integer=1, alpha="auto_po2")))
    model.add(QActivation(activation=quantized_relu(n_bits, integer=1), name="relu1"))

    return model
```

LISTING 1.

network parameters down to the 8-bit precision. In Listing 1, we reported the function that defines the NN model. For sake of conciseness, we only reported the first NN layer. The parameter n_{bits} corresponds to the bitwidth, defining the precision of the quantization. Additionally, within the function quantized_bits() the secondary parameter represents the number of bits allocated for the integer segment of the quantified value. For this instance, we selected 1 bit for the integer component, although alternative selections are viable. It should be noted that the integer bit excludes the sign bit **(1)**. The count of unsigned bits equals the total quantized number of bits minus the signed bit.

$$unsigned_{bit} = n_{bits} - signe\,d_{bit} = 8 - 1 \tag{1}$$

Furthermore, the fractional part of the fixed-point number equals the unsigned bits minus the integer bits.

The QKeras functions can be used in order to compile the model, fit it, and calculate the accuracy using the test set.

As explained in the paper, our contribution consisted in manipulating the trained weights and exporting them so that they could be included in the MCU code. The following step is then saving the quantized parameters in a way that is compatible with CMSIS. In order to do that, we changed the QKeras function *model_save_quantized_weight()* (see Listing 2). Specifically, the *quantizer()* outputs, as it usually returns a fixed-point quantized value, while in contrast CMSIS only works with integer quantized values. In this appendix we'll take the int8 quantization as a reference. In the function *save_parameters()* every parameter needs to be multiplied by an integer value mul, which in our case can be calculated as follows:

$$mul = \frac{2^{unsigned_{bit}}}{2^{integer}} = \frac{2^7}{2} = 64 \tag{2}$$

As an example, let's consider a quantization to 8 bits of a non-negative numbers, the lowest possible $number_{lowest} = 0$. The smallest fixed-point number is $number_{fixedpoint} = 1/2^8 = 1/256 = 0.00390625$. As we cannot work with fixed-point values when using CMSIS, we need the smallest number to be the smallest integer (aka. 1). To achieve this, we multiply $number_{fixedpoint}$ by mul, where mul equals to $2^{unsigned_{bit}}/2^{integer} = 2^8$. Or rewritten: $number_{integer} = number_{fixedpoint} * mul$.

The last step to make QKeras parameters compatible with CMSIS is to reshape the weights. The below code (Listing 3), adapted from an example that achieves this for convolutional layers,[4] transforms the weights into a CMSIS compatible format:

Now the parameters are ready to be used by CMSIS functions.

We now explain the full implementation to run a quantized network using CMSIS.

Below some additional considerations in case the provided code is wished to be extended:

Notation: We will use the Q notation used by ARM to represent a fixed-point number. This means that Qm.f is a fixed-point number that has m bits in the integer part of the value counting the sign bit, and f fractional bits.

In the CMSIS library, The data are assumed in dynamic fixed-point format. For example, a q7_t input number can be Q4.3, where the actual represented value is the int8 value divided by 2^3, but it can be Q1.7 as well, where the actual represented value is the int8 value divided by 2^7. As mentioned before, we'll take the int8 configuration as a reference example.

Let's assume that we have the input data in the Q1.7 format and the NN's parameters in the Q2.6. The first fully connected layer is taken as an example, where the output of the fully connected layer in chosen to be in the Q2.6 format. The definition of the ARM fully connected function is shown in Listing4, we'll later explain the usage of the parameters in bold.

We specify for completeness that our parameters are in the Q2.6 format because the value of "integer" in the QKeras *create_model()* function was chosen to be 1, this means that we have 2 bits in the integer part of the fixed-point number, and one of the two is the signed bit.

The core operation of the fully connected is the matrix multiplication between weight and input matrixes, the result is then summed to the bias matrix (W*I + B = OUT).

[4]https://developer.arm.com/documentation/102591/2208/Compare-the-ML-framework-and-CMSIS-NN-data-layouts

```
def save_parameters(model, mul):

    par_list = []
    # File name and location where the parameters will be saved to
    file = open("C:/My_folder/saved_param.txt", "w+")

    for layer in model.layers:
          # Some layers don't have quantizer ( ex.   Activation("softmax"))
        if hasattr(layer, "get_quantizers"):
            qs = layer.get_quantizers()
            ws = layer.get_weights()

            for quantizer, weight in zip(qs, ws):

                if quantizer:
                    weight = tf.constant(weight)
                    weight = tf.keras.backend.eval(quantizer(weight)) # Quantize parameters
                    weight = np.array(weight)
                    hw_weight = np.round(weight * mul)

                      # If hw_weight are weights (not bias) reshape them
                    if hw_weight.ndim > 1:
                            hw_weight = reshape_weights(hw_weight)

                    par_list.append(hw_weight)
                    file.write("\n\n" + layer.name + " ")
                    hw_weight.tofile(file, sep=", ", format="%d")

    file.close()
    return par_list
```

LISTING 2.

```
def reshape_weights(weights):
    transposed_wts = np.transpose(weights)
    return np.reshape(transposed_wts,
(transposed_wts.shape[0]*transposed_wts.shape[1]
))
```

LISTING 3.

```
arm_status   arm_fully_connected_q7(const   q7_t
*pV, const q7_t *pM, const uint16_t dim_vec,
        const    uint16_t    num_of_rows,    const
uint16_t bias_shift, const uint16_t out_shift,
        const   q7_t   *bias,   q7_t   *pOut,q15_t
*vec_buffer);
```

LISTING 4.

In fixed-point, W*I becomes Q2.6 * Q1.7, then the result is saved in a 32-bit accumulator, in the Q19.13 format.

The reason for this result is that in fixed-point multiplication, the number of fractional bits have to be summed to obtain the correct format. In this case, the result will have 6+7 fractional bits, and the rest of the 32-bit accumulator is assigned to the integer part (32-13=19). This is the reason why Q2.6 * Q1.7 = Q19.13.

Before summing the bias (Q2.6) to the multiplication result (Q19.13) these two addends need to be in the same format. For this purpose, we have to use the **bias_shift** parameter of the fully connected CMSIS function, bias_shift should be 7 since a right shift of 7 adjusts the difference between biases and the multiplication result.

The output layer format is Q19.13 as well. Since we chose a Q2.6 output format, a left shift of 7 is necessary for the output. Thus, the value 7 has to be assigned to the **out_shift** parameter of the fully connected CMSIS function.

These operations for the first layer become:

$$\boldsymbol{Q2.6\,(W)\,{}^{*}Q1.7\,(I)} + Q2.6 \ll 7\,(\boldsymbol{B}) = \boldsymbol{Q19.13} \gg \boldsymbol{7(OUT)}$$

The arm_fully_connected_q7() function doesn't only sum the bias to the multiplication result, it sums NN_ROUND(out_shift) together with the bias, where NN_ROUND(out_shift) equals $\boldsymbol{2^{out_shift-1}}$. Even if the function is meant to compensate for the error due to the n-bit shifting of the output, it alters the bias values, and it could introduce an error.

ACKNOWLEDGMENT

Max Ortiz-Catalan has been a consultant for an orthopedic implant company. Fabian Just, Chiara Ghinami, and Jan Zbinden report no conflict of interest.

(Fabian Just and Chiara Ghinami contributed equally to this work.)

REFERENCES

[1] M. Ortiz-Catalan, B. Håkansson, and R. Brånemark, "Real-time and simultaneous control of artificial limbs based on pattern recognition algorithms," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 22, no. 4, pp. 756–764, Jul. 2014, doi: 10.1109/TNSRE.2014.2305097.

[2] J. Zbinden, J. Molin, and M. Ortiz-Catalan, "Deep learning for enhanced prosthetic control: Real-time motor intent decoding for simultaneous control of artificial limbs," *IEEE Trans. Neural Syst. Rehabil. Eng.*, doi: 10.1109/TNSRE.2024.3371896.

[3] J. Zbinden, P. Sassu, E. Mastinu, E. J. Earley, M. Munoz-Novoa, R. Brånemark, and M. Ortiz-Catalan, "Improved control of a prosthetic limb by surgically creating electro-neuromuscular constructs with implanted electrodes," *Sci. Translational Med.*, vol. 15, no. 704, Jul. 2023, Art. no. eabq3665, doi: 10.1126/scitranslmed.abq3665.

[4] M. D. Paskett, M. R. Brinton, T. C. Hansen, J. A. George, T. S. Davis, C. C. Duncan, and G. A. Clark, "Activities of daily living with bionic arm improved by combination training and latching filter in prosthesis control comparison," *J. Neuroeng. Rehabil.*, vol. 18, no. 1, pp. 1–18, Dec. 2021, doi: 10.1186/s12984-021-00839-x.

[5] N. E. Krausz, D. Lamotte, I. Batzianoulis, L. J. Hargrove, S. Micera, and A. Billard, "Intent prediction based on biomechanical coordination of EMG and vision-filtered gaze for end-point control of an arm prosthesis," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 28, no. 6, pp. 1471–1480, Jun. 2020, doi: 10.1109/TNSRE.2020.2992885.

[6] N. Y. Sattar, Z. Kausar, S. A. Usama, U. Farooq, and U. S. Khan, "EMG based control of transhumeral prosthesis using machine learning algorithms," *Int. J. Control, Autom. Syst.*, vol. 19, no. 10, pp. 3522–3532, Oct. 2021, doi: 10.1007/s12555-019-1058-5.

[7] A. M. Simon, K. L. Turner, L. A. Miller, L. J. Hargrove, and T. A. Kuiken, "Pattern recognition and direct control home use of a multi-articulating hand prosthesis," in *Proc. IEEE Int. Conf. Rehabil. Robot.*, Jun. 2019, pp. 386–391, doi: 10.1109/ICORR.2019.8779539.

[8] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators," *Commun. ACM*, vol. 63, no. 7, pp. 48–57, Jun. 2020, doi: 10.1145/3361682.

[9] H. Han and J. Siebert, "TinyML: A systematic review and synthesis of existing research," in *Proc. Int. Conf. Artif. Intell. Inf. Commun. (ICAIIC)*, Feb. 2022, pp. 269–274, doi: 10.1109/ICAIIC54071.2022.9722636.

[10] I. L. Orăşan, C. Seiculescu, and C. D. Căleanu, "A brief review of deep neural network implementations for ARM cortex-M processor," *Electronics*, vol. 11, no. 16, p. 2545, Aug. 2022, doi: 10.3390/electronics11162545.

[11] L. Zhang, G. Liu, B. Han, Z. Wang, and T. Zhang, "SEMG based human motion intention recognition," *J. Robot.*, vol. 2019, pp. 1–12, Aug. 2019, doi: 10.1155/2019/3679174.

[12] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for modern deep learning research," in *Proc. 34th AAAI Conf. Artif. Intell.*, vol. 1, 2020, pp. 1393–13696, doi: 10.1609/aaai.v34i09.7123.

[13] L. Heim, A. Biri, Z. Qu, and L. Thiele, "Measuring what really matters: Optimizing neural networks for TinyML," 2021, *arXiv:2104.10645*.

[14] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent.*, Feb. 2016, pp. 1–14.

[15] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713, doi: 10.1109/CVPR.2018.00286.

[16] I. Hubara, M. Courbariaux, and D. Soudry, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, pp. 1–30, Jan. 2018.

[17] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proc. 33rd Int. Conf. Mach. Learn. (ICML)*, Nov. 2016, pp. 4166–4175.

[18] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, R. Rhodes, T. Wang, and P. Warden, "TensorFlow lite micro: Embedded machine learning on TinyML systems," 2020, *arXiv:2010.08678*.

[19] F. Just. (2023). *QNN MCU*. [Online]. Available: https://github.com/fabianjust/QNN-MCU

[20] M. R. Azghadi, C. Lammie, J. K. Eshraghian, M. Payvand, E. Donati, B. Linares-Barranco, and G. Indiveri, "Hardware implementation of deep network accelerators towards healthcare and biomedical applications," *IEEE Trans. Biomed. Circuits Syst.*, vol. 14, no. 6, pp. 1138–1159, Dec. 2020, doi: 10.1109/TBCAS.2020.3036081.

[21] D. Zhang, S. Mishra, E. Brynjolfsson, J. Etchemendy, D. Ganguli, B. Grosz, T. Lyons, J. Manyika, J. C. Niebles, M. Sellitto, Y. Shoham, J. Clark, and R. Perrault, "2021 AI Index Report," AI Index Steer. Committee, Stanford Univ. Human-Centered Artif. Intell. Inst., Stanford, CA, USA, Tech. Rep., 2021, pp. 1–222. [Online]. Available: https://aiindex.stanford.edu/report/, doi: 10.48550/arXiv.2103.06312.

[22] W. Jiang, X. Ye, R. Chen, F. Su, M. Lin, Y. Ma, Y. Zhu, and S. Huang, "Wearable on-device deep learning system for hand gesture recognition based on FPGA accelerator," *Math. Biosci. Eng.*, vol. 18, no. 1, pp. 132–153, 2021, doi: 10.3934/mbe.2021007.

[23] L. Cerina, G. Franco, P. Cancian, and M. D. Santambrogio, "Robustness of surface EMG classifiers with fixed-point decomposition on reconfigurable architecture," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2018, pp. 146–153, doi: 10.1109/IPDPSW.2018.00030.

[24] H. Wöhrle, M. Tabie, S. Kim, F. Kirchner, and E. Kirchner, "A hybrid FPGA-based system for EEG- and EMG-based online movement prediction," *Sensors*, vol. 17, no. 7, p. 1552, Jul. 2017, doi: 10.3390/s17071552.

[25] A. Boschmann, A. Agne, L. Witschen, G. Thombansen, F. Kraus, and M. Platzner, "FPGA-based acceleration of high density myoelectric signal processing," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2015, pp. 1–8, doi: 10.1109/ReConFig.2015.7393312.

[26] Z. Yang, A. B. Clark, D. Chappell, and N. Rojas, "Instinctive real-time sEMG-based control of prosthetic hand with reduced data acquisition and embedded deep learning training," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 5666–5672, doi: 10.1109/ICRA46639.2022.9811741.

[27] A. T. Nguyen, M. W. Drealan, D. Khue Luu, M. Jiang, J. Xu, J. Cheng, Q. Zhao, E. W. Keefer, and Z. Yang, "A portable, self-contained neuroprosthetic hand with deep learning-based finger control," *J. Neural Eng.*, vol. 18, no. 5, Oct. 2021, Art. no. 056051, doi: 10.1088/1741-2552/ac2a8d.

[28] S. Tam, M. Boukadoum, A. Campeau-Lecours, and B. Gosselin, "A fully embedded adaptive real-time hand gesture classifier leveraging HD-sEMG and deep learning," *IEEE Trans. Biomed. Circuits Syst.*, vol. 14, no. 2, pp. 232–243, Apr. 2020, doi: 10.1109/TBCAS.2019.2955641.

[29] A. Vitale, E. Donati, R. Germann, and M. Magno, "Neuromorphic edge computing for biomedical applications: Gesture classification using EMG signals," *IEEE Sensors J.*, vol. 22, no. 20, pp. 19490–19499, Oct. 2022, doi: 10.1109/JSEN.2022.3194678.

[30] E. Ceolini, C. Frenkel, S. B. Shrestha, G. Taverni, L. Khacef, M. Payvand, and E. Donati, "Hand-gesture recognition based on EMG and event-based camera sensor fusion: A benchmark in neuromorphic computing," *Frontiers Neurosci.*, vol. 14, pp. 1–15, Aug. 2020, doi: 10.3389/fnins.2020.00637.

[31] 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 Trans. Biomed. Circuits Syst.*, vol. 13, no. 5, pp. 795–803, Oct. 2019, doi: 10.1109/TBCAS.2019.2925454.

[32] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, "Opportunities for neuromorphic computing algorithms and applications," *Nature Comput. Sci.*, vol. 2, no. 1, pp. 10–19, Jan. 2022, doi: 10.1038/s43588-021-00184-y.

[33] 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," *Phil. Trans. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 378, no. 2164, Feb. 2020, Art. no. 20190155, doi: 10.1098/rsta.2019.0155.

[34] S. Raurale, J. McAllister, and J. M. del Rincon, "EMG wrist-hand motion recognition system for real-time embedded platform," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 1523–1527.

[35] S. Pancholi and A. M. Joshi, "Electromyography-based hand gesture recognition system for upper limb amputees," *IEEE Sensors Lett.*, vol. 3, no. 3, pp. 1–4, Mar. 2019, doi: 10.1109/LSENS.2019.2898257.

[36] E. Mastinu, P. Doguet, Y. Botquin, B. Håkansson, and M. Ortiz-Catalan, "Embedded system for prosthetic control using implanted neuromuscular interfaces accessed via an osseointegrated implant," *IEEE Trans. Biomed. Circuits Syst.*, vol. 11, no. 4, pp. 867–877, Aug. 2017, doi: 10.1109/TBCAS.2017.2694710.

[37] X. Liu, J. Sacks, M. Zhang, A. G. Richardson, T. H. Lucas, and J. Van der Spiegel, "The virtual trackpad: An electromyography-based, wireless, real-time, low-power, embedded hand-gesture-recognition system using an event-driven artificial neural network," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 11, pp. 1257–1261, Nov. 2017, doi: 10.1109/TCSII.2016.2635674.

[38] K. Xu, W. Guo, L. Hua, X. Sheng, and X. Zhu, "A prosthetic arm based on EMG pattern recognition," in *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, Dec. 2016, pp. 1179–1184, doi: 10.1109/ROBIO.2016.7866485.

[39] J. Liu, F. Zhang, and H. H. Huang, "An open and configurable embedded system for EMG pattern recognition implementation for artificial arms," in *Proc. 36th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, Aug. 2014, pp. 4095–4098, doi: 10.1109/EMBC.2014.6944524.

[40] Y. Guo, "A survey on methods and theories of quantized neural networks," 2018, *arXiv:1808.04752*.

[41] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018, *arXiv:1806.08342*.

[42] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 32, 2019, pp. 1–9.

[43] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by half-wave Gaussian quantization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5406–5414, doi: 10.1109/CVPR.2017.574.

[44] M. Courbariaux, Y. Bengio, and J.-P. David, "Low precision storage for deep learning," in *Proc. ICLR*, vol. 5, 2015, pp. 1–10, doi: 10.48550/arXiv.1511.00363.

[45] M. Courbariaux, Y. Bengio, and J. P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2015, pp. 3123–3131.

[46] R. Ding, T.-W. Chin, Z. Liu, and D. Marculescu, "Regularizing activation distribution for training binarized deep networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 11400–11409, doi: 10.1109/CVPR.2019.01167.

[47] Y. Guo, A. Yao, H. Zhao, and Y. Chen, "Network sketching: Exploiting binary structure in deep CNNs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4040–4048, doi: 10.1109/CVPR.2017.430.

[48] Y. Zhou, S. M. Moosavi-Dezfooli, N. M. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," in *Proc. 32nd AAAI Conf. Artif. Intell. AAAI*, 2018, pp. 4596–4604, doi: 10.1609/aaai.v32i1.11623.

[49] S. Khoram and J. Li, "Adaptive quantization of neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–13.

[50] S. A. Tailor, J. Fernandez-Marques, and N. D. Lane, "Degree-quant: Quantization-aware training for graph neural networks," in *Proc. 9th Int. Conf. Learn. Represent. (ICLR)*, 2021, pp. 1–22.

[51] H. D. Nguyen, A. Alexandridis, and A. Mouchtaris, "Quantization aware training with absolute-cosine regularization for automatic speech recognition," in *Proc. Annu. Conf. Int. Speech Commun. Assoc. Interspeech*, Oct. 2020, pp. 3366–3370, doi: 10.21437/interspeech.2020-1991.

[52] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer, "ZeroQ: A novel zero shot quantization framework," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 13166–13175, doi: 10.1109/CVPR42600.2020.01318.

[53] J. Fang, A. Shafiee, H. Abdel-Aziz, D. Thorsley, G. Georgiadis, and J. H. Hassoun, "Post-training piecewise linear quantization for deep neural networks," in *Proc. 16th Eur. Conf. Comput. Vis. (ECCV)*, Glasgow, U.K.: Springer, Aug. 2020, pp. 69–86.

[54] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*. London, U.K.: Chapman & Hall, 2022, pp. 291–326.

[55] B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst, "Minimum energy quantized neural networks," in *Proc. 51st Asilomar Conf. Signals, Syst., Comput.*, Oct. 2017, pp. 1921–1925, doi: 10.1109/ACSSC.2017.8335699.

[56] 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 Mach. Intell.*, vol. 3, no. 8, pp. 675–686, Jun. 2021, doi: 10.1038/s42256-021-00356-5.

[57] S. Lobov, N. Krilova, I. Kastalskiy, V. Kazantsev, and V. Makarov. (2019). *EMG Data for Gestures Data Set*. UCI Mach. Learing Repository. [Online]. Available: https://archive.ics.uci.edu/ml/machine-learning-databases/00481/

[58] A. Jaramillo-Yánez, M. E. Benalcázar, and E. Mena-Maldonado, "Real-time hand gesture recognition using surface electromyography and machine learning: A systematic literature review," *Sensors*, vol. 20, no. 9, p. 2467, Apr. 2020, doi: 10.3390/s20092467.

[59] M. Fabietti, M. Mahmud, and A. Lotfi, "On-chip machine learning for portable systems: Application to electroencephalography-based brain–computer interfaces," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–8, doi: 10.1109/IJCNN52387.2021.9533413.

[60] P.-E. Novac, G. Boukli Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, "Quantization and deployment of deep neural networks on microcontrollers," *Sensors*, vol. 21, no. 9, p. 2984, Apr. 2021, doi: 10.3390/s21092984.

[61] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for arm Cortex-M CPUs," 2018, *arXiv:1801.06601*.

[62] A. Skillman and T. Edsö, "A technical overview of cortex-M55 and ethos-u55: Arm's most capable processors for endpoint AI," in *Proc. IEEE Hot Chips 32 Symp. (HCS)*, Aug. 2020, pp. 1–20, doi: 10.1109/HCS49909.2020.9220415.

[63] E. J. Earley, J. Zbinden, M. Munoz-Novoa, E. Mastinu, A. Smiles, and M. Ortiz-Catalan, "Competitive motivation increased home use and improved prosthesis self-perception after cybathlon 2020 for neuromusculoskeletal prosthesis user," *J. Neuroeng. Rehabil.*, vol. 19, no. 1, pp. 1–11, Dec. 2022, doi: 10.1186/s12984-022-01024-4.

FABIAN JUST (Member, IEEE) received the M.Sc. degree in electrical engineering from Purdue University, IN, USA, in 2013, the M.Sc. degree in control engineering from Ruhr-University Bochum, Germany, in 2014, and the Ph.D. degree in rehabilitation robotics from ETH Zürich, Switzerland, in 2020. He is currently a Postdoctoral Researcher with the Center for Bionics and Pain Research, specializing in prosthetic and exoskeleton control. As the Founding Head-of-Discipline and an Advisory Board Member of the CYBATHLON, he fosters inclusion for people with disabilities.

CHIARA GHINAMI received the M.S. degree in embedded systems engineering from Cagliari University, in 2023. She is currently pursuing the Ph.D. degree in electronic engineering with RWTH University, Aachen, Germany. In July 2022, she joined the CBPR Research Laboratory, where she carried out the master's thesis on the deployment of neural networks on constrained hardware devices. She is currently working on hardware security for embedded systems.

JAN ZBINDEN (Graduate Student Member, IEEE) received the B.Sc. degree in mechanical engineering and the M.Sc. degree in mechanical engineering (robotics and rehabilitation engineering) from ETH Zürich, Switzerland, in 2017 and 2018, respectively. He is currently pursuing the Ph.D. degree in electrical engineering with the Chalmers University of Technology, Sweden, and the Center for Bionics and Pain Research, Sweden. His research interests include prosthetic control and prosthetic embodiment.

MAX ORTIZ-CATALAN (Senior Member, IEEE) is currently the Head of neural prosthetic research with the Bionics Institute, Melbourne, Australia, and an Honorary Principal Fellow with The University of Melbourne, Melbourne. He has authored over 100 scientific publications and has been a guest speaker at over 100 international conferences and universities worldwide. Several documentaries and over 100 popular science articles in over a dozen languages have featured his work, for which he has received several honors for innovation and scientific excellence.