SYNtzulu-Conv: Enabling Spiking 2D Convolutions for Sensor Data Analysis on Low-Power FPGAs

Gianluca Leone
Dipartimento di Ingegneria Elettrica ed Elettronica
Università degli Studi di Cagliari
Cagliari, CA, Italy
gianluca.leone94@unica.it

Federico Mura
Dipartimento di Ingegneria Elettrica ed Elettronica
Università degli Studi di Cagliari
Cagliari, CA, Italy
f.mura60@studenti.unica.it

Luigi Raffo
Dipartimento di Ingegneria Elettrica ed Elettronica
Università degli Studi di Cagliari
Cagliari, CA, Italy
raffo@unica.it

Paolo Meloni
Dipartimento di Ingegneria Elettrica ed Elettronica
Università degli Studi di Cagliari
Cagliari, CA, Italy
paolo.meloni@unica.it

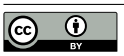
Abstract

Spiking Neural Networks (SNNs) are energy-efficient algorithms that have proven effective for AI-based sensor-data processing at the edge. The new generation of high-density sensor arrays enables augmented decoding and classification capabilities for sensing devices, however, similarly to what has been observed in image processing, dense layers become impractical as the sensor array size increases, both due to the growing number of channels, which increases the memory footprint of AI models, and because they fail to capture spatial features inherently present when sensor sensing areas overlap. This paper introduces a new end-to-end spiking convolutional accelerator, designed for deployment on low-cost and low-power FPGA devices for high-count sensor data analysis. The system integrates a RISC-V softcore responsible for managing input/output dataflow, an encoding and a decoding slot for sample-to-spike data conversion and viceversa, and a programmable and sparsity-aware convolutional spiking neural network accelerator. We evaluated the system, implemented on a Lattice iCE40UP5K FPGA, on a High-Density surface EMG continuous force tracking task included in the Hyser dataset. The proposed solution achieves accuracy comparable to that of a dense baseline model, while reducing the memory footprint by a factor of 5.1× and dissipating only 35 pJ per synaptic operation, corresponding to 6.8 mW during inference.

Keywords

Spiking Neural Networks, FPGA, Convolutions, EMG, low-power

ACM Reference Format:
Gianluca Leone, Federico Mura, Luigi Raffo, and Paolo Meloni. 2026. SYNtzulu-Conv: Enabling Spiking 2D Convolutions for Sensor Data Analysis on Low-Power FPGAs. In *Proceedings of the 16th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART 2026), June 17–19, 2026, Heidelberg, Germany*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3814576.3814595



HEART 2026, Heidelberg, Germany

ACM ISBN 979-8-4007-2452-7/26/06
https://doi.org/10.1145/3814576.3814595

1 Introduction

Spiking Neural Networks (SNNs) are receiving growing attention in AI-based sensor data processing, due to a number of intrinsic computational properties. First, both inter-layer and input/output data are represented as single-bit events, namely spikes, which simplify the computation of neurons' input currents by replacing dense multiply-and-accumulate operations, typical of vanilla neural networks, with sparse additions. Second, the internal state of spiking neurons naturally provides temporal awareness, making these models well suited for processing time-series of data. Neuromorphic processors can efficiently exploit the computational advantages of spiking workloads [21, 24, 26], however, they cannot yet be regarded as a mainstream solution because of their cost and limited availability. A compelling alternative for the execution of SNNs is represented by FPGA devices, which provide flexible memory resources, DSP capabilities, and reconfigurable logic that can be tailored to embed sparsity-aware mechanisms within the data-crunching logic. This is especially relevant for continuous monitoring, where only relevant features captured by the sensors are translated into spikes [2, 4], natively reducing the workload when no relevant activity is present, and therefore, allowing sparsity-aware systems to save power during idle periods.

The resulting efficiency gains are even more significant when new-generation high-density sensor arrays are considered, where the growing number of sensing channels and the increased sampling rates intensify the computational and memory demands at the edge. In this scenario, scalability concerns also influence the choice of the model architecture. As in image processing, dense layers become impractical as the sensor array size increases, making support for convolutional layers necessary to enable lightweight feature-extraction stages preceding classification [9, 19, 33], while also allowing the capture of spatial features inherently present when sensor sensing areas overlap [22, 31].

Several FPGA-based accelerators for SNN execution have been proposed in the literature [1, 3, 5–8, 14–18, 23, 25, 27, 32, 34–36]. However, most of them are not specifically tailored to sensor data processing and instead primarily target bulk image classification on high-end device, with limited attention to edge deployment power constraint requirements. In this work, we build on SYNtzulu

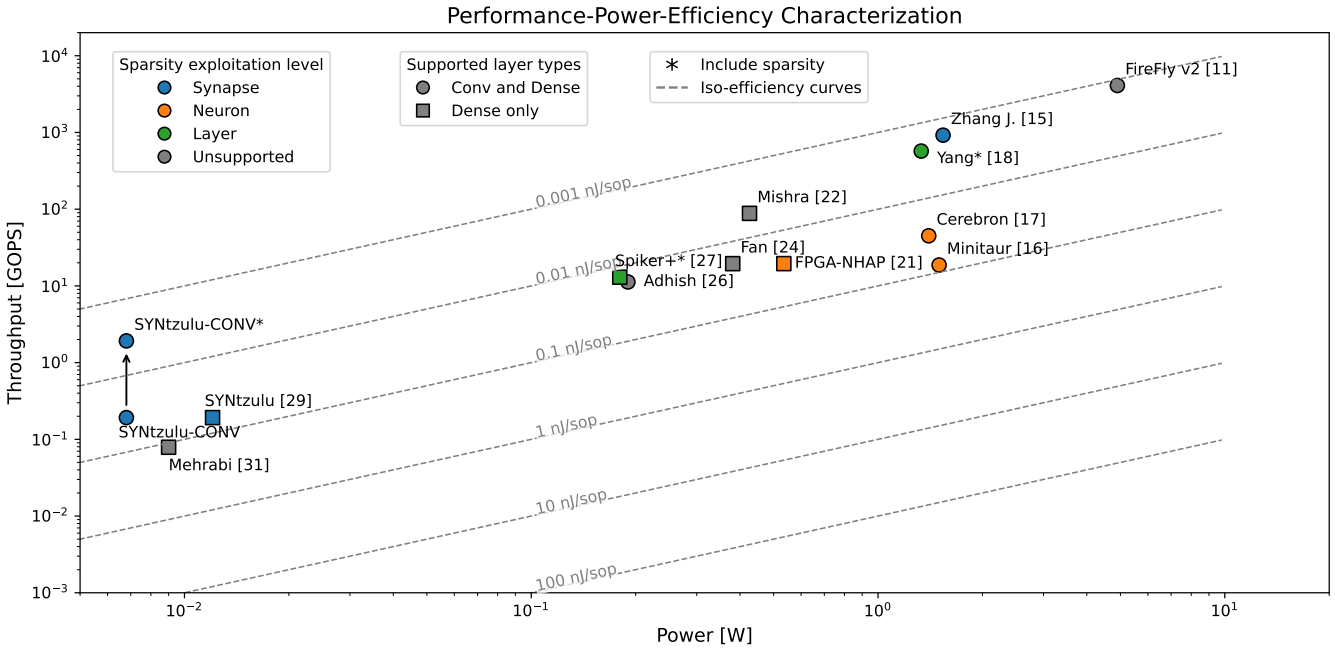


Figure 1: Overview of FPGA-based SNN accelerators.

[13], an existing SNN accelerator specifically developed to enable AI-based analysis at ultra-low power cost. Rather than maximizing throughput, SYNtzulu is designed to prioritize resource minimization and low-power operation, allowing deployment on low-power FPGA devices and making it particularly well suited to stringent edge-power constraints. We extend this architecture by adding support for sparsity-aware execution of convolutional spiking layers, thereby enabling the processing of high-density sensor arrays at the edge. The main contributions of this paper can be summarized as follows:

- we present a novel lightweight architectural approach to exploit sparsity in tightly resource-constrained spiking convolutional accelerators;
- we extend an existing FPGA-based SNN accelerator tailored for time-series analysis by integrating convolutional support, bridging the gap between high-density sensor processing at the edge and state-of-the-art FPGA-based SNN processors;
- in these scenarios, a 1:1 encoding of input variations into spikes is preferred over absolute-value frame encoding, where each input is typically expanded into multiple frames. This requires sparsity to be exploitable at a finer grain, to skip as many operations as possible in every single convolution kernel. SYNtzulu-Conv supports kernel-level sparsity control at the cost of few hundreds of logic cells.

2 State of the Art

Interest in FPGA-based spiking accelerators has increased markedly in recent years. The number of papers grew from an average of about 16 per year over the 2016–2022 period to 55 in 2025[1]. Figure 1 reports a selection of the most relevant works, highlighting the performance[2]/power trade-off and the energy efficiency of the accelerators, along with other features such as support for dense and convolutional layers, or for dense layers only, as well as the type of sparsity exploited, if any.

Sparsity can be exploited at different levels of granularity. For example, SYNtzulu [13] and [35] exploit sparsity at the synaptic level, meaning that during inference the weights associated with inactive synapses are not fetched from memory. However, neuron-state integration is still performed at every time step, since these designs rely on a single-step Euler approximation and therefore require continuous integration. This approach in literature is usually referred as clock-driven. Other accelerators, such as [7, 18, 25], follow a fully event-driven paradigm, skipping not only inactive synapses but also neuron updates when no input spikes are received. This approach generally requires a more complex neuron integration scheme, including the computation of an exponential term. Which solution is more convenient depends on the sparsity level and on how frequently neuron integration must be performed [16].

Finally, other designs exploit sparsity at a much coarser granularity, skipping only entire layers [6, 34]. While this simplifies the architecture, it may overlook significant performance gains. Indeed, accelerators that exploit sparsity can, depending on the sparsity of the target application, move upward in the plot of Figure 1,

[1]The counts were obtained from Google Scholar by restricting the search to article titles and publication year. The query used was `(intitle:SNN OR intitle:"spiking neural network" OR intitle:"spiking neural networks" OR intitle:neuromorphic) intitle:FPGA`, followed by the year filter in the left sidebar (e.g., custom range 2025–2025).

[2]The throughput values reported for the works marked with an asterisk include the effect of computation skipping enabled by sparsity.

achieving substantially higher throughput and energy efficiency. For example, [5] reports that, in a rate-encoded MNIST application, only 23 frames out of 3,500 were active, resulting in a 152× speed-up. Similarly, [29] measured a data sparsity of 92% in an ECG classification task, leading to a 12.5× gain. This suggests that accelerators supporting sparsity can potentially increase both throughput and energy efficiency by up to 1 to 2 orders of magnitude. For reference, Figure 1 also reports the efficiency of our architecture under 90% sparsity, marked with an asterisk. The dashed lines represent iso-efficiency curves in the design space, allowing architectures with different power and throughput characteristics to be directly compared in terms of energy efficiency.

Most of the works in Figure 1 are concentrated on the right-hand side of the plot, showing that the vast majority of FPGA-based SNN accelerators consume at least several hundreds of milliwatts. This is consistent with the typical quiescent power consumption of off-the-shelf FPGA devices. Moreover, the overall trend indicates that accelerator energy efficiency improves with increasing throughput. Indeed, higher parallelism helps mitigate the impact of quiescent power and reduces the relative contribution of control overhead. As a result, the most energy-efficient designs are also those achieving the highest throughput[3] [14, 35], and they target high-end FPGA platforms such as Kintex UltraScale+ FPGA and Zynq UltraScale+ MPSoC. However, ultra-high parallelism makes it more difficult to accommodate a non-uniform computation paradigm; consequently, FireFly v2 [14] does not support sparsity.

This trend also highlights the difficulty of designing energy-efficient accelerators under the very tight power budget typical of near-sensor processing. In this respect, this work, together with [20] and SYNtzulu [13], lies on the far left side of Figure 1, representing one of the few viable options for edge AI applications constrained to around 10 mW. As discussed above, supporting sparsity can increase throughput and energy efficiency consistently depending on the characteristics of the input data and the adopted SNN model. Therefore, in realistic deployment scenarios, the gap between [20] and both this work and SYNtzulu [13] is expected to become even more pronounced, since they can shift upward in the plot by skipping inactive synaptic computations. Finally, while the proposed architecture achieves the same throughput as SYNtzulu and relies on the same sparsity-exploitation strategy, it extends this capability to convolutional layers as well. Supporting convolutional layers in addition to dense layers within such a limited power envelope is essential to enable real-time AI processing of high-density sensor arrays at the edge.

The contribution of this work compared to the state of the art are:

- the proposed lightweight approach enables implementing 2D convolutions on low-power devices, decreasing significantly active and idle power figures and enabling a ultra-edge exploitation;
- this solution is usable and beneficial for near-sensor processing of 2D organized sensor data;
- in these scenarios, a 1:1 encoding of input variations into spikes is preferred over absolute-value frame encoding, where each input is typically expanded into multiple frames. This requires sparsity to be exploitable at a finer grain, to skip as many operations as possible within single convolution kernels. SYNtzulu-Conv supports kernel-level sparsity control at the cost of few hundreds of logic cells.

[3]The work [34] is omitted since its throughput account for the skipped computations due to sparsity exploitation.

3 Hardware Architecture

The design proposed in this work builds upon the open-source SNN accelerator[4] SYNtzulu [13], a compact FPGA-based system designed for real-time near-sensor processing under strict power constraints. Starting from this baseline, we extend the architecture with support for spiking convolutional layers while preserving sparsity-aware execution and remaining within the tight resource budget of the target Lattice iCE40UP5K FPGA.

3.1 Sytem Overview

The architecture, shown in Fig. 2, is implemented on a Lattice iCE40UP5K FPGA, a compact low-power device featuring 4 256-kb single-port RAMs (SPRAMs), 30 4-kb dual-port block RAMs (BRAMs), 8 DSPs, and 5280 logic cells (LCs), each comprising a 4-input LUT and one flip-flop, and two oscillators.

The system integrates a SErial RISC-V (SERV) core, featuring an ultra-compact bit-serial architecture. Due to its limited computational throughput, the core is not involved in SNN inference or signal-processing tasks. Instead, it provides a flexible control layer, enabling rapid system reconfiguration across different applications through firmware updates compiled with a standard RISC-V toolchain. SERV manages memory-mapped peripherals via load/store operations, including SPI and UART communication (for input streaming and output transmission), power management, and configuration of system parameters.

At the core of the system lies a programmable SNN accelerator, whose execution is controlled through a dedicated instruction memory. Each custom instruction configures the computation associated with a specific network layer, allowing different SNN topologies to be mapped onto the same hardware without requiring architectural modifications.

The SNN accelerator is interfaced with the external environment through configurable encoding and decoding slots, which adapt the processor to application-specific data formats and output requirements. The encoding slot converts continuous sensor data into spike trains, timely associating spikes to input variations, while the decoding slot interprets the generated spikes into usable outputs (e.g., regression values or classification results). The decoding slot is typically simpler and, in some cases, can be omitted. For regression tasks, neuron membrane potentials can be directly used as outputs. In contrast, classification tasks require spike-based decision logic, such as spike counting over a time window followed by winner selection.

After initialization, the middleware coordinates system operation through a timer-driven interrupt mechanism, periodically triggering the acquisition of new input data streams. Once the inputs are encoded, inference is performed, and the outputs are decoded, the system transitions into a light standby state by disabling the high-frequency oscillator. The system timer is instead driven by a secondary 10 kHz oscillator. While the system must operate at

[4]https://github.com/gianlucaleone/SYNtzulu

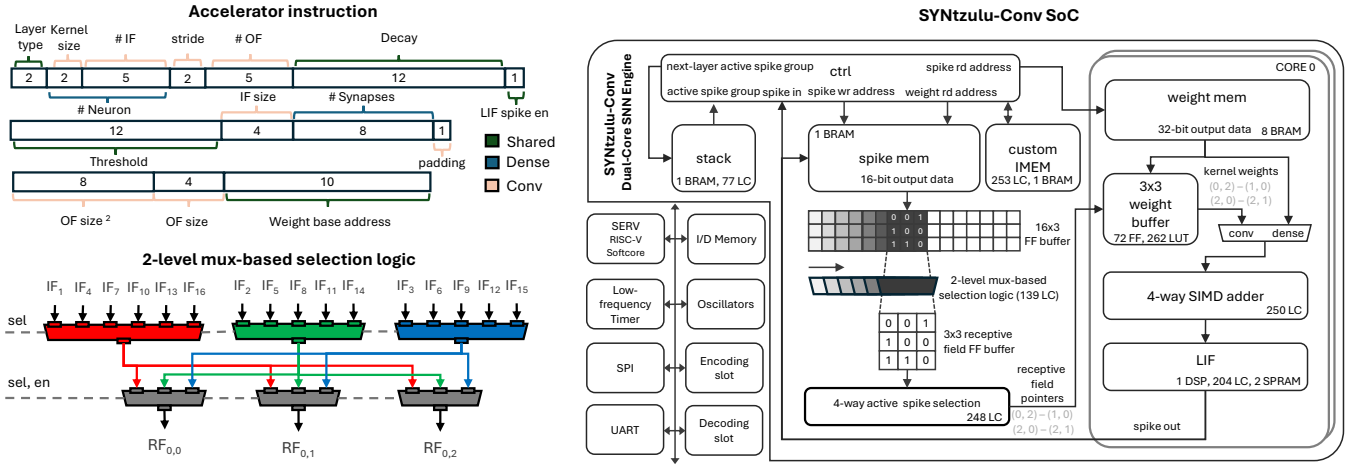


Figure 2: In the top-left corner, the custom accelerator instruction format is shown; on the right, the system block diagram; and in the bottom-left corner, the multiplexer-based receptive field extraction logic.

a sufficiently high clock frequency to meet real-time constraints in the worst-case scenario (i.e., in the absence of sparsity), this approach enables the conversion of time savings, achieved under typical sparse conditions, into reduced power consumption.

3.2 Preliminary Dense Architecture

The baseline architecture [13] is designed to execute feedforward SNNs composed of dense layers of LIF neurons, as in Equation 1.

$$\begin{aligned} V(t) &= \alpha V(t-1) + I(t), \\ I(t) &= \sum w_i s_i(t), \\ s(t) &= V(t) > \theta, \\ V(t) &= [1 - s(t)]V(t) + s(t)[V(t) - \theta] \end{aligned} \tag{1}$$

Where, $V(t)$ is the neuron membrane potential, α is the potential decay, $I(t)$ is the neuron input current, computed as the scalar product between the activation of the input synapses $s_i(t)$ and the weights of the input synapses w_i, θ is the firing threshold above which a spike is fired and the neuron potential is reset, and $s(t)$ is the neuron output spike.

The SNN accelerator consists of two parallel cores, each including a 4-way SIMD adder and a LIF integration unit. The workload is evenly distributed between the two cores, each processing half of the neurons in every layer. Synaptic weights dominate the memory footprint in dense models and are therefore stored in the on-chip SPRAM blocks, which provide the largest memory resources available on the device. Neuron membrane potentials, in contrast, are mapped onto multiple BRAM blocks. Spikes are packed in groups of four and stored in a single BRAM-based buffer, which temporarily holds neuron outputs for subsequent processing.

A key feature of the baseline design is sparsity-aware execution. During the LIF integration of layer *i*, an active-spike stack records the active spike sets addresses and dynamically schedules synaptic weights processing for layer *i+1*, so that only active synapse sets are evaluated. This reduces the number of operations and increases idle time as a function of input sparsity.

3.3 Convolutional Architecture Extension

The baseline architecture described above targets dense feedforward SNNs and already provides efficient sparsity-aware execution on a compact FPGA device. However, high-density sensor applications often require convolutional processing to capture local spatial features while keeping the model memory footprint limited for edge deployment. To address this need, the proposed architecture extends the baseline design with support for spiking convolutional layers, as shown in Figure 2.

In this work, we support 3×3 and 2×2 convolutional kernels, as larger kernels are generally more suitable for high-resolution image processing than for high-density sensors, which typically feature only a few hundred recording sites. The architecture supports convolution strides of 1, 2, and 3, as well as 2×2 max-pooling, and padding. Moreover, since in convolutional SNNs the memory distribution shifts from synaptic weights to neuron membrane potentials, we store neuron potentials in SPRAMs, allowing support for up to 32,768 membrane states, while BRAMs are used for weight storage. In this implementation, 16 BRAM blocks are reserved for weights, corresponding to a total capacity of 8,192 weights. Finally, to improve programmability and reusability, we integrated an instruction memory within the accelerator. Each instruction spans 5 BRAM words and is used to configure the accelerator, with each instruction corresponding to a specific network layer.

The accelerator was extended to support convolutional layer execution with resource efficiency as a primary design objective. To this end, most of the datapath components already used for dense computation were reused during convolutional execution. In particular, each core still processes four synaptic weights per cycle, allowing the existing 4-way SIMD adder to be reused, while the same LIF integration module is also shared.

To maintain a lightweight architecture, the computation of output features is evenly partitioned across the two cores. This enables the use of a unified control logic for input feature reading and extraction, with input features stored sequentially in the global spike

memory, while each core stores its own convolutional and dense weights in a private weight memory.

3.3.1 Custom Instructions and Instruction Memory. To enable straightforward reuse of the core across different applications and network topologies, we employ a BRAM to store 75-bit custom instructions that define the topology to be executed. Each instruction includes a 2-bit layer-type opcode specifying whether the operation corresponds to a dense, convolutional, pooling, or convolution-plus-dense layer; the latter option also triggers the automatic flattening of the output feature maps. The remaining instruction bits encode the relevant layer parameters, such as the number and size of the input and output features, and whether stride and padding are enabled. For dense layers, part of this encoding is interpreted differently, representing instead the number of input synapses and output neurons. Finally, some parameters are common to all layer types, including the base address of the weights in memory, the neuron membrane potential decay, and the firing threshold.

3.3.2 Convolutional operation. To enable efficient access during convolution, three lines of flip-flops are instantiated to buffer three lines of input feature at a time. Similarly, inside each core, a 3×3 buffer is used to allow fast access to the kernel weights. To extract the 3×3 receptive field is used a simple selection logic composed of two stages of multiplexers, shown to the bottom-left side of Figure 2. The receptive field is stored into a 3×3 matrix of flip-flops.
Sparsity in convolutional layers is exploited within each receptive field. A dedicated logic composed of four custom priority encoders extracts up to four active spike pointers per clock cycle. These pointers are then used to dynamically schedule kernel-weight accumulation through the 4-way SIMD adder. When more than four elements are active within the receptive field, the flip-flops storing the currently fetched entries are cleared, allowing up to four additional weights to be scheduled in the following cycle. In the worst-case scenario, when all 9 elements in the 3×3 receptive field are active, the convolution requires 3 clock cycles.
The output of each convolution contributes to the input current of a LIF neuron. The total current of each neuron is obtained as the sum of the contributions from all input features, as expressed in Equation 2. Since the input features are processed sequentially, storing and accumulating all partial currents would require an additional buffer. To avoid this overhead, we slightly modified the LIF neuron module by introducing an external control mechanism that independently enables membrane potential decay, current addition, and spike condition evaluation.

$$\begin{aligned} V(t) &= \alpha V(t-1) + I(t), \\ I(t) &= \sum_{k=0}^{N} I_{\mathrm{IF}_i}(t) \end{aligned} \tag{2}$$

With this approach, when the first input feature is processed, the membrane potential is decayed and the first current contribution is added. For all subsequent input features except the last one, only current accumulation is performed. Finally, once the contribution of the last input feature has been added, the spike condition is evaluated. This scheme eliminates the need for an additional buffer to store intermediate neuron currents.

3.3.3 Stride. Stride is implemented through the receptive-field selection logic. Since the selection is realized with multiplexers rather than with line buffers shifted by one position at a time, supporting stride values of one, two, or three does not introduce additional complexity.

3.3.4 Padding. Padding is implemented by leveraging the receptive-field selection logic. Specifically, the second stage of multiplexers can be selectively enabled or disabled, with disabled multiplexers forcing their outputs to zero. This simple mechanism allows zero-padding of the top and bottom borders of the input feature map by disabling all multiplexers, and of the left and right borders by disabling only the first or last multiplexer.

3.3.5 Max Pooling. The max-pooling operation reuses most of the control logic and buffering structures employed for convolutions. Once the receptive field buffer has been loaded, the priority encoders check whether at least one active spike is present. If so, the max-pooling output is set to one, and the result is written directly into the spike memory.

4 High-Density sEMG Continuous Tracking Task

We evaluated the proposed system on a continuous finger force tracking task based on high-density surface electromyography (HD-sEMG) signals. This application represents a challenging real-time regression problem, requiring the extraction of fine-grained spatio-temporal features from multi-channel biopotentials under strict latency and power constraints.

4.1 Reference Dataset

The evaluation is conducted using the Hyser (High-densitY Surface Electromyogram Recordings) dataset [10, 11], a publicly available benchmark comprising recordings from 20 subjects. The dataset includes multiple acquisition protocols, covering gesture recognition and force estimation tasks with synchronized HD-sEMG and finger force measurements. In this work, we focus on the subset dedicated to single-degree-of-freedom finger contractions, where each finger is independently activated while the others remain relaxed. The task consists of tracking continuous force trajectories with a triangular profile, where each subject performs flexion and extension contractions reaching 30% of their maximum voluntary contraction (MVC). Each subject performs multiple trials across two recording sessions.
The sEMG signals are acquired using four high-density electrode grids placed on the forearm, for a total of 256 channels, sampled at 2048 Hz. Force measurements are recorded synchronously at 100 Hz. Following the procedures described in [11, 12], sEMG signals are low-pass filtered and downsampled to 1000 Hz, while force signals are filtered (10 Hz cutoff) and resampled to the same rate. Experiments are conducted in a single-session setting, using data from session 2 for both training and testing, consistent with prior works on this dataset [12, 28].

4.2 HD-sEMG Encoding

To reduce memory footprint and simplify the encoding logic, the input sEMG signals are first quantized to 8-bit resolution. This

choice enables efficient hardware implementation while preserving sufficient signal fidelity for the target regression task. Spike generation is performed using a delta modulation scheme, which encodes temporal variations of the signal rather than its absolute value. In this approach, a spike is emitted whenever the difference between the current input sample and a running reference exceeds a predefined threshold Δ. The reference is then updated accordingly, enabling a compact and event-driven representation of the signal dynamics. Formally, given an input signal $x(t)$ and a reconstructed signal $\hat{x}(t)$, the encoding operates by comparing their difference against a threshold Δ:

- if $x(t) - \hat{x}(t) > \Delta$, a positive spike is generated and $\hat{x}(t)$ is increased of Δ;
- if $x(t) - \hat{x}(t) < -\Delta$, a negative spike is generated and $\hat{x}(t)$ is decreased of Δ;
- otherwise, no spike is emitted.

The value of Δ was selected independently for each channel by maximizing the Pearson correlation coefficient between the running sum of the delta-modulated signal and the corresponding finger force traces. To this end, for each sEMG channel, Δ was explored over 20 values spanning from 1/30 to 1/3 of the peak value measured on the training set, and the value yielding the highest correlation was retained.

4.3 SNN Topology, Training, and Quantization

The SNN model is designed to efficiently process high-dimensional spatio-temporal inputs while remaining compatible with the hardware constraints of the proposed architecture. The network topology is detailed in Tables 1 and 2, together with the main training parameters. Training is performed offline using the SNN-Torch framework. The model is trained to minimize the error between the membrane potentials of the five output neurons, i.e. one per finger, and the ground-truth force signals. Therefore, we set the threshold in the last layer to 1e9 to prevent spike firing, while it is set to 0.9 in the other layers and kept non-trainable. The neuron decay is set to 0.9 and is trainable at the layer level.

Table 1: SNN topology

Layer	Input dimensions	Weights	Neurons
Conv1	$16 \times 16 \times 2$	288	3136
Conv2	$14 \times 14 \times 16$	1152	1568
Pool1	$14 \times 14 \times 8$	–	–
Conv3	$7 \times 7 \times 8$	1152	400
Dense	400	2000	5
Total		4592	5109

After training, the model weights were quantized to 8 bits and the neuron membrane potential to 16 bits, resulting in an accuracy drop of only 0.06%. State observers were used to verify that no overflow occurred during LIF integration.

5 Results

This section evaluates the proposed system across hardware efficiency and application performance. We first report FPGA implementation results in terms of resource utilization, throughput, and power consumption. We then assess the accuracy on the HD-sEMG continuous force tracking task. Finally, we compare the proposed convolutional network approach with a dense baseline model.

Table 2: Training setup

Hyperparameter	Value
Optimizer	Adam
Learning rate	3e-4
Batch size	190
Epochs	800
Loss function	MSE
Time steps	500 ms

5.1 Resource Usage and Power Consumption

Logic synthesis and place-and-route were performed using an architecture-neutral open-source FPGA toolchain[5], relying on Yosys Open SYnthesis Suite for logic synthesis and nextpnr for placement, routing, and bitstream generation [30]. The resulting resource utilization is summarized in Table 3. Logic cell usage reached 86.3%.

	LCs	BRAM	SPRAM	DSP
Used	4557	30	4	2
Utilization	86.3%	100%	100%	25%

Table 3: FPGA resource utilization.

BRAM utilization reached 100%, with memory resources allocated as follows: 16 BRAMs for synaptic weight storage, 8 BRAMs for the softcore instruction/data memory, 2 BRAMs for spike memory, 1 BRAMs for stack memory, 1 BRAMs for the encoding slot, 1 BRAM for the accelerator output buffer, and 1 BRAM for the accelerator instruction memory. In addition, 2 DSP blocks are used within the dual-core accelerator for neuron state integration.

The system was clocked at 24 MHz. Its throughput, equal to 0.192 GOPS (10^9 operations per second), is obtained by multiplying the clock frequency by eight, since the architecture employs two cores, each capable of processing four synapses per clock cycle in both dense and convolutional layers, while neuron integration is overlapped with synaptic current evaluation. Power consumption was measured during both the active inference phase and the standby state. Measurements were obtained by sensing the voltage drop across a shunt resistor with nominal value $3.3 \pm 0.033\ \Omega$, soldered on the VCORE supply rail of the iCEBreaker V1.0e prototype board. The average power consumption measured during inference is 6.8 mW, and 0.21 mW in standby.

5.2 HD-sEMG Accuracy Assessment

The reported results are obtained under a one-day, leave-one-repetition-out evaluation protocol. A three-fold cross-validation scheme is adopted, where each of the three trials within a session is alternately used as the test set, while the remaining two are used for training. Performance is evaluated by comparing the network outputs with the ground-truth force signals, reporting the average results across the three folds in terms of Pearson correlation coefficient (CC).

[5] https://github.com/yosyshq/oss-cad-suite-build

Table 4: Comparison with the SoA

	Work	Data	FPGA	Clock/Event	Sparsity	Encoding	Convolutions	Inference power	mW/MHz	Energy/synapse [nJ]
	FireFly v2 [14]	image	Zynq UltraScale+ MPSoC	clock-driven	–	none	√	4.9 W	9.8***	0.0012
	FPSpike [15]	image	Zynq UltraScale+ MPSoC	clock-driven	structured	frame-based	√	3.15 W	3.15	-
	UIC [36]	image	Zynq UltraScale+ MPSoC	event-driven	neuron	frame-based	×	2.97 W	19.8	-
	Li Et al. [16]	image	Virtex-7	both	neuron	frame-based	×	1.6 W	16	28
> 1 W	Zhang J. Et al. [35]	image	Zynq UltraScale+ MPSoC	clock-driven	structured	frame-based	√	1.54 W	7.7	0.0017
	Minitaur [25]	image	Spartan-6	event-driven	neuron	frame-based	×	1.5 W	20	-
	Cerebron [7]	image	Zynq-7000 SoC	clock-driven*	synapse	–	√	1.4 W	7	0.031
	Yang Et Al. [34]	image	Zynq UltraScale+ MPSoC	clock-driven	synapse	frame-based	√	1.332 W	13.32	–
	DeepFire2[3]	image	Virtex UltraScale+	clock-driven	–	frame-based	√	–	–	0.014
	Cactus [32]	image/speech	Kintex UltraScale	clock-driven	structured & synapse	frame-based	√	–	–	0.018
	FPGA-NHAP [18]	image	Kintex-7	event-driven	neuron	frame-based	×	535 mW	2.675	27.4
	Mishra Et al. [23]	image	Artix-7	clock-driven	–	frame-based	×	426 mW	4.26	0.5
	SyncNN[27]	image	Zynq UltraScale+ MPSoC	event-driven	neuron	frame-based	√	400 mW	2	-
< 1 W,	Fan Et al. [8]	image	Artix-7	clock-driven	–	none	×	381 mW	3.81	1.95
> 100 mW	Liu Et al.[17]	image	Zynq-7000 SoC	clock-driven	synapse**	frame-based	√	280 mW	2.8	-
	Adhish [1]	image	Artix-7	clock-driven	–	frame-based	×	190 mW	1.9	–
	Spiker+ [6]	image/speech	Zynq-7000 SoC	clock-driven	layer	frame-based	×	180 mW	1.8	1.37
	Spiker [5]	image	Artix-7	clock-driven	layer	frame-based	×	–	–	41
	Mehrabi Et al.[20]	sensors	Cyclone V	clock-driven	–	event-based	×	9 mW	3.75	–
< 100 mW	SYNtzulu [13]	sensors	iCE40UP5K	clock-driven	synapse	event-based	×	12.05 mW	0.54	0.062
	SYNtzulu-Conv	sensors	iCE40UP5K	clock-driven	synapse	event-based	√	6.8 mW	0.28	0.035 (3.5)
				No difference with IF neurons*	Only partially exploited**				Considering 500 MHz***	

The proposed model is trained as a global predictor, capable of processing HD-sEMG data from multiple subjects[6] and simultaneously estimating the force exerted by each finger. Quantitative results show an average Pearson correlation coefficient of 0.914. After quantization, using 8-bit precision for the weights and 16-bit precision for neuron membrane potentials, we observed a negligible accuracy degradation of only 0.06%. Furthermore, as shown in Figure 3, the float32 and quantized inference outputs are visually well aligned to the ground truth, confirming the ability of the model to accurately track continuous force profiles. As a reference, we also trained a dense model under the same training conditions. Its topology consists of fully connected layers of size $64 \times 128 \times 64 \times 5$, with 512 input channels corresponding to the flattened $16 \times 16 \times 2$ input obtained using the same encoding scheme adopted in this work. Overall, the dense topology comprises a total of 49.7k trainable parameters, compared with 9.7k for the proposed convolutional model. This baseline achieved a Pearson correlation coefficient of 0.932. Figure 4 provides a visual interpretation of this difference in correlation coefficient: although the dense model achieves slightly higher accuracy, the predictions of the proposed model remain closely aligned with the ground-truth force profiles. Finally, we evaluated the sparsity of the algorithm on the recording corresponding to subject 1, session 2, for finger 1, as well as the effectiveness of our architecture in exploiting it. We measured an average algorithmic sparsity of 96.64%. Since the architecture performs at least four parallel additions, due to the adoption of a 4-way SIMD adder, some loss in exploitable sparsity is expected. In particular, whenever between one and four spikes are present within the kernel window, the hardware still activates a 4-input adder. As a result, the measured hardware sparsity is 55.5%.

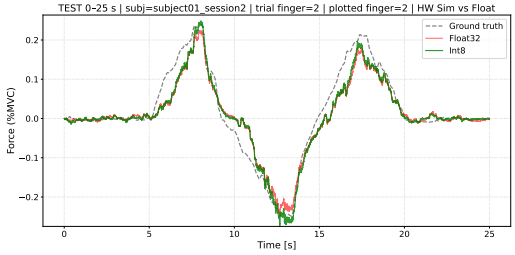


Figure 3: Continuous force tracking: Float32 vs Int8.

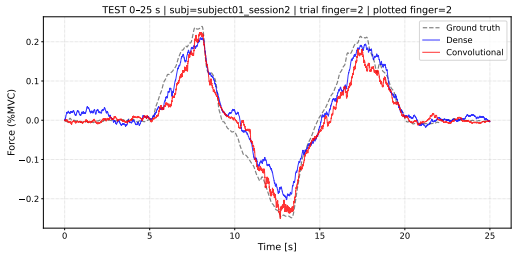


Figure 4: Continuous force tracking: Convolutional vs Dense

6 Comparison with the State of the Art

Table 4 reports the works discussed qualitatively in Section 2, along with additional works that were excluded from the previous analysis due to the lack of necessary data for plotting, such as throughput in GOPS or absolute power consumption in watts. As discussed in Section 2, this work is the only viable FPGA solution supporting convolutional SNN execution within the power envelope of near-sensor processing. The works listed in Table 4 can be grouped into three clusters: architectures consuming more than 1 W, between 100 mW and 1 W, and below 100 mW. Only three architectures fall into the last group. Among them, this work is the most efficient, and the only one that supports convolutional layers while also exploiting data sparsity to skip unnecessary computations.

In terms of energy efficiency, this work achieves the best result within its cluster, consuming 0.035 nJ per synaptic operation, i.e., 41.5% less than [13]. To allow a fair comparison with the work in the two high-power clusters, in terms of energy per synapse,

[6]Patient 20 was excluded, as it appears to be mislabeled.

it must be considered that the overall synaptic workload for the rate-encoded images corresponds to multiple time steps (usually 100, 25 for [1]). Moreover, sparsity must be accounted for, when exploitable. Thus, we show a second figure in the table, equal to 3.5 nJ, presenting the energy dissipated by SYNtzulu-Conv over 100 time steps. Under these considerations, the proposed architecture is competitive with designs in the mid-power range, achieving 7.8× and 11.7× lower energy per synaptic operation compared to [18] and [5], respectively, even without accounting for sparsity exploitation in our estimates. However, [23] remains more efficient, requiring only 14.3% of the energy per synaptic operation of our approach. Even when compared with architectures operating above 1 W, this work remains 8× more efficient than [16]. Other works, such as Cactus [32] and DeepFire2 [3] achieve higher efficiency. However, their absolute power consumption is not reported. Notably, Xilinx Power Estimator 2023.2.1 reports a static power of 1.183 W for the Kintex UltraScale FPGA used in [32] (XCKU115), and 0.773 W for the Virtex UltraScale FPGA considered for DeepFire2 (XCZU9EG[7]). These values alone are already 174× and 114× higher than the active power consumption of our design, respectively, thus placing Cactus and DeepFire2, as well as the other architectures in their cluster, well outside the power envelope of near-sensor processing. Finally, with respect to frequency-normalized power, this work dissipates only 0.28 mW/MHz during inference, corresponding to a 48% reduction with respect to SYNtzulu and to at least a 6.4× improvement over the best result among the related works, achieved by [6], highlighting the hardware efficiency of the presented architecture. Overall, these results demonstrate that sparsity-aware convolutional SNN execution can be achieved within ultra-low-power regimes, a target not addressed by existing FPGA-based solutions.

7 Conclusion

In this work, we presented a lightweight SNN accelerator tailored for ultra-low-power near-sensor processing, extending the open-source SYNtzulu architecture with support for sparsity-aware convolutional layers. The proposed design preserves the key strengths of the baseline system, i.e. resource minimization and energy-efficient execution, while enabling a broader class of portable applications targeting high-density sensor array utilization.

The architecture has been implemented on a Lattice iCE40UP5K FPGA, demonstrating that convolutional SNN execution can be achieved within 6.8 mW. Experimental results on a real-world task, namely real-time finger-force decoding from high-density surface EMG signals, show that the proposed approach reduces the memory footprint by 5.1× with respect to a dense topology, while maintaining comparable accuracy.

Compared to the state of the art, SYNtzulu-Conv emerges as the only viable solution for executing convolutional spiking models under the strict power constraints of near-sensor processing applications, showing that sparsity-aware convolutional SNNs can be efficiently deployed on ultra-low-power FPGA platforms and thus bridging the gap between advanced neural processing capabilities and the stringent constraints of edge and near-sensor applications.

[7]The exact device is not reported in [3]; therefore, we conservatively selected the smallest device in the family.

Acknowledgments

This work was supported by European Union's Horizion 2020 research and Innovation program under the grand agreement No. GA 101140052 (H2TRAIN).

References

[1] JS Adhish, Vijaya Kumar, and Suresh Balanethiram. 2025. An Efficient FPGA Implementation of Spiking Neural Networks for N-MNIST Classification. In *2025 IEEE 7th International Conference on Emerging Electronics (ICEE)*. IEEE, 1–4.

[2] D. Auge, J. Hille, E. Mueller, and A. Knoll. 2021. A Survey of Encoding Techniques for Signal Processing in Spiking Neural Networks. *Neural Processing Letters* 53 (2021), 4693–4710. doi:10.1007/s11063-021-10562-2

[3] Myat Thu Linn Aung, Daniel Gerlinghoff, Chuping Qu, Liwei Yang, Tian Huang, Rick Siow Mong Goh, Tao Luo, and Weng-Fai Wong. 2023. Deepfire2: A convolutional spiking neural network accelerator on fpgas. *IEEE Trans. Comput.* 72, 10 (2023), 2847–2857.

[4] Paola Busia, Gianluca Leone, Andrea Matticola, Luigi Raffo, and Paolo Meloni. 2025. Wearable epilepsy seizure detection on fpga with spiking neural networks. *IEEE Transactions on Biomedical Circuits and Systems* (2025).

[5] Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. 2022. Spiker: an fpga-optimized hardware accelerator for spiking neural networks. In *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 14–19.

[6] Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. 2024. Spiker+: a framework for the generation of efficient Spiking Neural Networks FPGA accelerators for inference at the edge. *IEEE Transactions on Emerging Topics in Computing* 13, 3 (2024), 784–798.

[7] Qinyu Chen, Chang Gao, and Yuxiang Fu. 2022. Cerebron: A reconfigurable architecture for spatiotemporal sparse spiking neural networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30, 10 (2022), 1425–1437.

[8] Andrew Fan and Simon D Levy. 2025. A Robust, Open-Source Framework for Spiking Neural Networks on Low-End FPGAs. *arXiv preprint arXiv:2507.07284* (2025).

[9] Jun Sung Go and Jong Tae Kim. 2026. Partitioned Convolutional Spiking Neural Network for Tactile Object Recognition. *Neural Processing Letters* 58, 1 (6 Jan. 2026), 11. doi:10.1007/s11063-025-11822-1

[10] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. 2000. PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. *circulation* 101, 23 (2000), e215–e220.

[11] Xinyu Jiang, Xiangyu Liu, Jiahao Fan, Xinming Ye, Chenyun Dai, Edward A Clancy, Metin Akay, and Wei Chen. 2021. Open access dataset, toolbox and benchmark processing results of high-density surface electromyogram recordings. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 29 (2021), 1035–1046.

[12] Xinyu Jiang, Kianoush Nazarpour, and Chenyun Dai. 2023. Explainable and robust deep forests for EMG-force modeling. *IEEE Journal of Biomedical and Health Informatics* 27, 6 (2023), 2841–2852.

[13] Gianluca Leone, Matteo Antonio Scrugli, Lorenzo Badas, Luca Martis, Luigi Raffo, and Paolo Meloni. 2025. SYNtzulu: A Tiny RISC-V-Controlled SNN Processor for Real-Time Sensor Data Analysis on Low-Power FPGAs. *IEEE Transactions on Circuits and Systems I: Regular Papers* 72, 2 (2025), 790–801. doi:10.1109/TCSI.2024.3450966

[14] Jindong Li, Guobin Shen, Dongcheng Zhao, Qian Zhang, and Yi Zeng. 2024. Firefly v2: Advancing hardware support for high-performance spiking neural network with a spatiotemporal fpga accelerator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 9 (2024), 2647–2660.

[15] Mingyang Li, Yirong Kan, Man Wu, Renyuan Zhang, and Yasuhiko Nakashima. 2026. FPSpike: A Fully Parallel and Reconfigurable Architecture for Accelerating Spiking Neural Networks With Structured Sparsity. *IEEE Transactions on Circuits and Systems I: Regular Papers* (2026).

[16] Sixu Li, Zhaomin Zhang, Ruixin Mao, Jianbiao Xiao, Liang Chang, and Jun Zhou. 2021. A fast and energy-efficient SNN processor with adaptive clock/event-driven computation scheme and online learning. *IEEE Transactions on Circuits and Systems I: Regular Papers* 68, 4 (2021), 1543–1552.

[17] Hanwen Liu, Yi Chen, Zihang Zeng, Malu Zhang, and Hong Qu. 2023. A Low Power and Low Latency FPGA-Based Spiking Neural Network Accelerator. In *2023 International Joint Conference on Neural Networks*. 1–8. doi:10.1109/IJCNN54540.2023.10191153

[18] Yijun Liu, Yuehai Chen, Wujian Ye, and Yu Gui. 2022. FPGA-NHAP: A general FPGA-based neuromorphic hardware acceleration platform with high speed and low power. *IEEE Transactions on Circuits and Systems I: Regular Papers* 69, 6 (2022), 2553–2566.

[19] Nathan Lutes, Venkata Sriram Siddhardh Nadendla, and K. Krishnamurthy. 2024. Convolutional spiking neural networks for intent detection based on anticipatory

brain potentials using electroencephalogram. *Scientific Reports* 14, 1 (17 April 2024), 8850. doi:10.1038/s41598-024-59469-7

[20] Ali Mehrabi and Gaetano Gargiulo. 2026. Hardware-Based Spiking Neural Network for Real-Time Hand Gesture Recognition Using MYO Armband EMG Sensor. *IEEE Transactions on Consumer Electronics* (2026), 1–1. doi:10.1109/TCE.2026.3658531

[21] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 6197 (2014), 668–673.

[22] Luca M Meyer, Majid Zamani, János Rokai, and Andreas Demosthenous. 2024. Deep learning-based spike sorting: a survey. *Journal of Neural Engineering* 21, 6 (2024), 061003.

[23] Saras Mani Mishra, Hanumant Singh Shekhawat, Jan Pidanic, and Gaurav Trivedi. 2025. FPGA-based adaptive LIF neuron for high-speed, energy-efficient spiking neural network. *IEEE Transactions on Circuits and Systems for Artificial Intelligence* (2025).

[24] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. 2017. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). *IEEE transactions on biomedical circuits and systems* 12, 1 (2017), 106–122.

[25] Daniel Neil and Shih-Chii Liu. 2014. Minitaur, an event-driven FPGA-based spiking network accelerator. *IEEE transactions on very large scale integration (VLSI) systems* 22, 12 (2014), 2621–2628.

[26] Garrick Orchard, E Paxon Frady, Daniel Ben Dayan Rubin, Sophia Sanborn, Sumit Bam Shrestha, Friedrich T Sommer, and Mike Davies. 2021. Efficient neuromorphic signal processing with loihi 2. In *2021 IEEE workshop on signal processing systems (SiPS)*. IEEE, 254–259.

[27] Sathish Panchapakesan, Zhenman Fang, and Jian Li. 2022. SyncNN: Evaluating and accelerating spiking neural networks on FPGAs. *ACM Transactions on Reconfigurable Technology and Systems* 15, 4 (2022), 1–27.

[28] Matteo Antonio Scrugli, Gianluca Leone, Paola Busia, Luigi Raffo, and Paolo Meloni. 2024. Real-time sEMG processing with spiking neural networks on a low-power 5K-LUT FPGA. *IEEE Transactions on Biomedical Circuits and Systems* 19, 1 (2024), 68–81.

[29] Matteo Antonio Scrugli, Gianluca Leone, Paola Busia, Luigi Raffo, and Paolo Meloni. 2026. All-Spiking ECG Analysis for Arrhythmia Classification on Low-Power FPGA. *IEEE Sensors Journal* (2026).

[30] David Shah, Eddie Hung, Clifford Wolf, Serge Bazanski, Dan Gisselquist, and Miodrag Milanovic. 2019. Yosys+ nextpnr: an open source framework from verilog to bitstream for commercial fpgas. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 1–4.

[31] Giacomo Valli, Paul Ritsche, Andrea Casolo, Francesco Negro, and Giuseppe De Vito. 2024. Tutorial: Analysis of central and peripheral motor unit properties from decomposed High-Density surface EMG signals with openhdemg. *Journal of Electromyography and Kinesiology* 74 (2024), 102850. doi:10.1016/j.jelekin.2023.102850

[32] Zilin Wang, Zehong Ou, Yi Zhong, and Yuan Wang. 2025. Cactus: A Multicore Spiking Neural Network Accelerator With Fine-Grained Structured Weight Sparsity. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2025).

[33] Mengjuan Xu, Xiang Chen, Antong Sun, Xu Zhang, and Xun Chen. 2023. A Novel Event-Driven Spiking Convolutional Neural Network for Electromyography Pattern Recognition. *IEEE Transactions on Biomedical Engineering* 70, 9 (2023), 2604–2615. doi:10.1109/TBME.2023.3258606

[34] Yuxuan Yang, Qihu Xie, Zihao Xuan, Song Chen, and Yi Kang. 2025. A neuromorphic hardware architecture based on TTFS coding with temporal quantization for spiking neural networks. *Integration* 103 (2025), 102403.

[35] Jian Zhang, Yong Wang, Yimao Cai, Bo Bi, Qiliang Chen, and Yanlong Zhang. 2025. A Low Power Spiking Neural Network Accelerator on FPGA for Real-Time Edge Computing. In *2025 Joint International Conference on Automation-Intelligence-Safety (ICAIS) & International Symposium on Autonomous Systems (ISAS)*. IEEE, 1–6.

[36] Qiang Zhang, Mingyue Cui, Weichong Chen, Yue Liu, and Zhiyi Yu. 2025. UIC: A unified and scalable chip integrating neuromorphic computation and general purpose processor. *Microelectronics Journal* 155 (2025), 106449.