

Spiking Neural Network-Based Near-Sensor Computing for Damage Detection in Structural Health Monitoring

Original

Spiking Neural Network-Based Near-Sensor Computing for Damage Detection in Structural Health Monitoring / Barchi, F; Zanatta, L; Parisi, E; Burrello, A; Brunelli, D; Bartolini, A; Acquaviva, A. - In: FUTURE INTERNET. - ISSN 1999-5903. - 13:8(2021). [10.3390/fi13080219]

Availability:

This version is available at: 11583/2978472 since: 2023-05-12T16:39:42Z

Publisher:

MDPI

Published

DOI:10.3390/fi13080219


Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

Spiking Neural Network based Near-Sensor Computing for Damage Detection in Structural Health Monitoring

Francesco Barchi ^{1,†,*} , Luca Zanatta ^{1,†}, Emanuele Parisi ^{1,†}, Alessio Burrello ¹, Davide Brunelli ², Andrea Bartolini¹ and Andrea Acquaviva ¹

¹ DEI, Università di Bologna

² DII, Università di Trento

* Correspondence: francesco.barchi@unibo.it

† These authors contributed equally to this work.

Abstract: In this work, we present an innovative approach for damage detection of infrastructures on edge devices, exploiting a brain-inspired algorithm. The proposed solution exploits recurrent Spiking Neural Network (LSNN), which are emerging for their theoretical energy efficiency and compactness, to recognise damage conditions by processing data from low-cost accelerometers (MEMS) directly on the sensor node. We focus on designing an efficient coding of MEMS data to optimise SNN execution on a low-power microcontroller. We characterised and profiled LSNN performance and energy consumption on a hardware prototype sensor node equipped with an STM32 embedded microcontroller and a digital MEMS accelerometer. We used a Hardware-in-the-Loop environment with virtual sensors generating data on an SPI interface connected to the physical microcontroller to evaluate the system with a data stream from a real viaduct. We exploited this environment also to study the impact of different on-sensor encoding techniques, mimicking a bio-inspired sensor able to generate events instead of accelerations. Obtained results show that the proposed optimised embedded LSNN (eLSNN), when using a spike-based input encoding technique, achieves 54% lower execution time with respect to a naive LSNN algorithm implementation present in the state-of-art. The optimised eLSNN requires around 47 kCycles, which is comparable with the data transfer cost from the SPI interface. However, the spike-based encoding technique requires considerably larger input vectors to get the same classification accuracy, resulting in a longer pre-processing and sensor access time. Overall the event-based encoding techniques leads to a longer execution time (1.49x) but similar energy consumption. Moving this coding on the sensor can remove this limitation leading to an overall more energy-efficient monitoring system.

Keywords: Spiking NN, SHM, Cyber-Physical Systems, Energy Efficiency, MEMS

Citation: Barchi, F.; Zanatta, L.; Parisi, E. et al. Spiking Neural Network based Near-Sensor Computing for Damage Detection in Structural Health Monitoring. *Future Internet* **2021**, *1*, 0. <https://doi.org/>

Received:

Accepted:

Published:

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2023 by the authors. Submitted to *Future Internet* for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The application of distributed sensors to pervasive monitoring of physical processes is one of the most critical and relevant domains. In particular, Structural Health Monitoring (SHM) is a key deployment scenario, where ensuring the safety of infrastructures such as buildings and bridges can be, in principle, achieved by deploying low-cost sensors to detect structural variations due to damages. The convergence of Artificial Intelligence (AI) to Internet-of-Things (IoT) and edge computing in this domain can help approach these challenges. In this context, executing AI detection algorithms directly on the IoT sensor nodes potentially reduces data transmission overheads and improves response time. A review of recent embedded AI approaches to detection in SHM can be found in [1]. Due to the low-cost, low-power and increasing accuracy of MEMS accelerometers, their application to distributed SHM is becoming popular.

Signal compression techniques on edge have also been proposed to compress MEMS data gathered from several nodes and sending them to the cloud storage and

analytic facility [2]. Still, on MEMS data, on-sensor modal estimation was proposed by implementing procedures to detect relevant peaks in the acquired signal spectrum [3].

Optimized machine learning algorithms on edge have been recently proposed to increase the intelligence of distributed detection for SHM. Considering the availability of code and libraries to implement machine learning algorithms on low-power microcontrollers [4,5], it is a viable solution to execute detection algorithms on-edge and on-sensor. In this context, hazard monitoring based on an array of event-triggered single-channel micro-seismic sensors with advanced signal processing is proposed exploiting a Convolutional Neural Network (CNN) implemented on a low-power microcontroller can be found in [6].

Applied to anomaly detection on a highway bridge, in [7] a compression technique to identify anomalies in the structure using a semi-supervised approach is proposed using either a fully connected or a convolutional autoencoder implemented on the sensor node. In the present work we provide an alternative solution applied to the same case study using a supervised algorithm for near-sensor anomaly detection based on Spiking Neural Networks (SNN). SNNs gained interest in the research community in various application domains, including SHM, because of their brain-inspired, event-based nature, which potentially allows reduced energy requirement compared to traditional ANN [8–12]. While Artificial Neural Networks (ANNs) have been successfully applied to SHM [1, 13–16], SNN are of increasing interest in this field because of their theoretical information greater processing efficiency by exploiting a sparse computation approach. In [17] a feed forward SNNs has been applied to low-cost, MEMS-based inspection of damaged buildings.

However, the state-of-art in SNN applications to SHM misses a real implementation of a data processing pipeline with the execution of SNN directly on the sensor node. Also, embedded machine learning libraries currently lack efficient SNN implementations. In the context of SNN, when time-series data from sensors are concerned, recurrent neural networks have shown to be effective [18]. For this reason, instead of more simple feed-forward architectures, we investigate recurrent SNNs for SHM data processing. In particular we consider a state-of-art recurrent implementation of SNN called LSNN (Long Short-Term SNN) introduced in [19] because of its interesting signal processing features and learning effectiveness. Also, a relevant aspect to be explored is the encoding of the input signal, which impacts subsequent computation steps and associated energy consumption. In particular, SNNs have been used with event-based input such as pixel variations from Dynamic Vision Sensor (DVS) cameras [20], but they can also effectively process "continuous" data streams in speech recognition applications [19]. However, in the context of SHM in general, which encoding is the best suited for anomaly detection task has not been studied so far.

This work presents the design, implementation, and characterisation of an LSNN on a low-power sensor node equipped with a commercial microcontroller and a MEMS accelerometer. The LSNN has been evaluated using real data from a highway viaduct, for which it was able to detect structural variations associated with a degraded condition. To the best of our knowledge, this is the first implementation of an LSNN on a low-power microcontroller that we integrated into a complete SNN-based near-sensor computing system. We designed and compared different input data encoding schemes in terms of performance and energy. We designed an optimised LSNN version for microcontroller targets, and we characterised its performance and energy consumption on silicon, including the overhead of data transfer from the MEMS sensor and their coding. Thanks to our Hardware-in-the-Loop measurement set-up, we were also able to emulate the behaviour of a smart sensor able to send spikes directly instead of acceleration values.

We are comparing with an alternative semi-supervised edge anomaly detection applied to the same dataset [7]. Authors of [7] show that the anomaly detection of the faulty and normal condition requires a complex pipeline. It consists of: i) A filtering step; ii) An anomaly detector; iii) A final smoothing post-processing. They either propose a

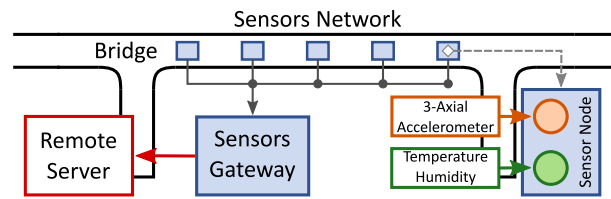


Figure 1. Monitoring system installation.

91 Principal Component Analysis compression and decompression or a fully connected
 92 autoencoder to implement the anomaly detector. The autoencoder features a single
 93 hidden layer of 32 neurons, an input layer with 500 samples, and an output layer with
 94 500 neurons. Both these algorithms show a complexity of $500 \times 32 \times 2$ (32 000) multiply-
 95 and-accumulate operations¹. We show that we can achieve similar accuracy results with
 96 considerably fewer computational resources, to solve the same detection problem, the
 97 proposed solution uses 15 750 sums and 750 multiplications.

98 While the two algorithms are not directly comparable because of the different
 99 ML approaches, the comparison against the reference testify that proposed solution is
 100 effective in solving the same detection problem. The contribution of this paper can be
 101 summarized as follows:

- 102 • We studied the computational requirements and complexity of the LSNN, and we
 103 provide an implementation on a low-power microcontroller-based sensor node.
- 104 • We designed and implemented an optimized LSNN version for performance con-
 105 strained architectures, and we compared a continuous versus event-based input
 106 encoding.
- 107 • We evaluated the benefits of the proposed optimizations both theoretically and on
 108 real hardware, and we explored accuracy versus energy and performance trade-offs,
 109 including the cost of sensor data transfer.
- 110 • We demonstrate that LSNNs can be effectively executed near the MEMS sensor
 111 with a few tens of K cycles (comparable with data transfer cost) and deliver MCC
 112 levels higher than 0.75 (corresponding to almost 90% of accuracy) using data from
 113 a real case study of damage detection in SHM.

114 The rest of this paper is organized as follows. Section 2 presents some background
 115 on SHM and LSNN. Section 3 presents the LSNN architecture, training and input
 116 coding methods. Section 4 explains the introduced optimizations. Section 5 reports the
 117 experimental test-bed and results, and Section 6 concludes the work.

118 2. Background

119 This section describes the SHM problem and the reference SoA monitoring system
 120 composed of sensor nodes, edge-node, and cloud architecture. We then give some
 121 background about Spiking Neural Networks and their recurrent LSNN counterparts,
 122 input coding strategy, and training algorithm adopted. Finally, the sensor board and
 123 microcontroller used for measurements are introduced.

124 As described earlier, the manuscript focuses on the feasibility analysis of using
 125 a brain-inspired algorithm on the sensor-node MicroController Unit (MCU) and its
 126 implementation trade-offs. To experimentally validate this approach in Section 5 we will
 127 describe the experimental setup consisting of a hardware-in-the-loop (HIL) approach.

128 2.1. Bridge Structure & SHM Framework

129 The structure under study is a highway viaduct ² built with eighteen sections,
 130 each one supported by two pairs of concrete pillars situated at their two ends. We

¹ The neurons, intended as the application in the hidden and output layers of a non-linear function like sigmoid or RELU, are 532 (hidden plus output neurons). Instead, the number of MAC operations is due to the number of connections. In this way, we have 500×32 MAC for the input-hidden layer and 32×500 MAC for the hidden-output layer, for a total of $500 \times 32 \times 2$ MAC.

² A32 Torino-Bardonecchia - Viadotto S.S.335

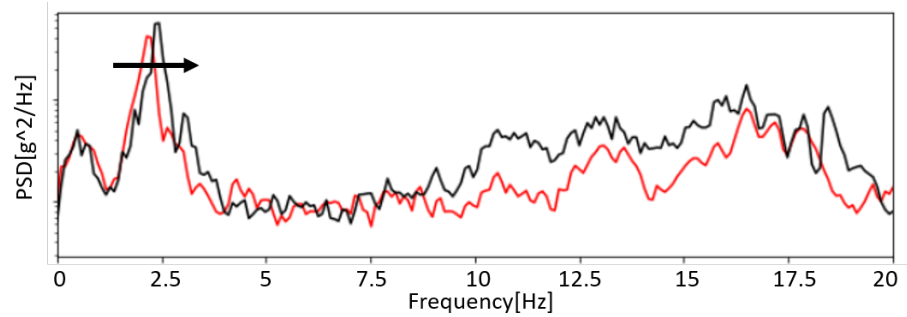


Figure 2. Mean natural frequency shift before and after scheduled maintenance.

131 focus on a single section instrumented for data analysis before a scheduled maintenance
 132 intervention in this work. Maintenance was necessary for the strengthening of the
 133 viaduct structure.

134 The acquisition framework is described in Figure 1. The system contains five
 135 identical sensor nodes. Each one features an STM32F4 microcontroller (MCU) and
 136 samples 3-axis accelerations and temperature data, and stores them into the cloud
 137 through a 4G-connected Raspberry Pi3 gateway.

138 The five nodes are connected through CAN-BUS to the gateway. Note that the
 139 acquisition system only collects the data without any local edge-side signal processing.
 140 In this basic configuration, data analysis is executed on the cloud. The accelerometer
 141 samples the data with a frequency of 25.6 kHz to avoid aliasing. Subsequently, the data
 142 is subsampled to have a final output frequency (f_s) of 100 Hz.

143 The cloud part is composed of a data-ingestion job, which receives and store data
 144 from the gateway, and periodically scheduled analysis tasks to monitor the health status
 145 of the bridge [21].

146 The MCU is an ARM 32-bit Cortex-M4 running at 168 MHz, with 192 kB of SRAM
 147 and 1 MB of Flash memory, popular in different edge applications for its low power
 148 consumption. Further, the MCU features a floating-point unit and a digital signal
 149 processing (DSP) library. The gateway is a standard Raspberry Pi 3 module B [22] (RPi3).
 150 It includes a Broadcom BCM2837 SoC, with 64 bits 4-core Cortex-A53 running at 1.2 GHz
 151 and 1 GB of DDR2 RAM. The gateway runs an Ubuntu operating system, easing the
 152 scheduling of communication tasks through common python interfaces (e.g., an MQTT
 153 broker [23]). The cloud system is divided into a storage section and a computing node
 154 allocated on the IBM cloud service.

155 In this work, we propose to replace cloud processing with a brain-inspired near-
 156 sensor anomaly detection algorithm directly computed on the microcontroller (MCU)
 157 on the sensor node board. We will show that the proposed algorithm can identify the
 158 normal and faulty bridge conditions, avoiding the RAW data transmission to the cloud.

159 2.2. Structural Health Monitoring Data

160 SHM frameworks usually process acceleration data for monitoring structures'
 161 health status [24]. The datasets used in the study contain 3-axial accelerations data
 162 acquired with the previously described framework. As above-mentioned, the viaduct
 163 underwent a technical intervention to strengthen its structure, with a corresponding
 164 change in the natural frequencies of a bridge. Figure 2 shows how the power spectrum
 165 density, averaged over 6 hours, is modified before and after the intervention.

166 Given these data's uniqueness, we use them as a proxy of an aged viaduct compared
 167 to a healthy one. In particular, in this work, we consider the signals collected after the
 168 intervention as the normal data produced by a healthy viaduct. Analogously, the data
 169 gathered before the maintenance intervention are considered "anomalies" since they are
 170 sampled on a damaged and aged bridge. Although the data do not represent the whole

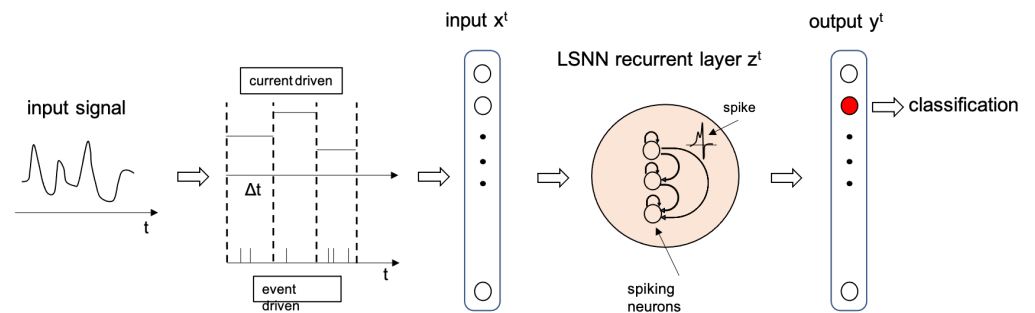


Figure 3. Full pipeline. The signal over time is encoded in current or events. Once transformed, it will be given as input to the SNN which will classify the data as a healthy or damaged bridge.

171 history of the viaduct, to the best of our knowledge, it is the only dataset containing
 172 vibrations from the viaduct during two different structural phases of the building.

173 2.3. Spiking Neural Networks

174 This paper proposes to study a Spiking Neural Network Model to solve the SHM
 175 supervised anomaly detection problem directly on the sensor's node MCU. The SNN
 176 model is a brain-inspired (so-called third-generation) type of neural network. They have
 177 a greater computational capacity as the single neuron is modelled with a much more
 178 complex dynamic than the neurons present in traditional Artificial Neural Networks
 179 (ANNs). This means that SNNs can solve the same tasks as ANNs with fewer neurons [8].
 180 Moreover, their hardware implementation on neuromorphic architectures and accelera-
 181 tors can lead to greater energy efficiency in data management and computation [25–29].
 182 In this work, we do not consider neuromorphic implementation because the objective is
 183 to work with low-cost commercial MCUs, for which SNN porting is not available.

184 In particular, we consider a recurrent type of SNN called LSNN (Long Short-Term
 185 SNN) because they are suitable to process temporal data streams like their artificial coun-
 186 terparts (e.g. LSTMs). While SNN has already attracted attention for SHM applications,
 187 so far, literature papers focused on simple feed-forward SNN, which are less powerful
 188 and do not exploit the potential of SNN, nor do they impose training challenges [17].

189 The recurrent LSNN structure is depicted in Figure 3, where the input, output and
 190 recurrent layers are represented. The input is a signal while the output is a classification
 191 encoded in the output neurons, meaning that each output neuron represents one of the
 192 possible classes. For instance, the neurons generating the highest output values is the
 193 one representing the recognised class. The time it takes to the network to process every
 194 single input and produce a stable output is called inference time (t_{inf}). As explained
 195 later in this Section, the input to the network can be either of current or event type. A
 196 current input type is a constant value over a time $\Delta t = t_{inf}$, while an event input type is
 197 a train of spikes, encoding the input signal using one of the possible methods described
 198 in Section 2.4.

199 This is the first work to study the feasibility of leveraging SNN inference on the
 200 sensor node on an embedded microcontroller. Detecting the health status of the structure
 201 directly on the node's MCU has the clear benefit of ease the network communication
 202 requirements of the sensors node to the edge node leading to energy reduction op-
 203 portunities. The event-driven nature of the SNN processing can lead to an optimised
 204 implementation consisting of (i) a coding of the input sensor stream into a sequence of
 205 events depending on the intensity of inputs and (ii) a computation workload (internal
 206 activity of the SNN) which processes these events as spikes. Since spikes are binary
 207 signals, linear algebra operators can be implemented with simplified arithmetic. We
 208 designed a pre-processing stage to apply LSNN to SHM real-life dataset. The state-of-
 209 the-art of LSNN applied to a similar problem of phonemes recognition is solved in [19],

210 which processes the TIMIT dataset (representing phonemes) by Mel-frequency cepstrum
211 (MFC). The spectral coefficients are given as input to the network as synaptic currents.

212 Starting from this reference LSNN (designed for server machines), we implemented
213 an LSNN to process the spectral coefficient of the accelerometer waveform to detect
214 a structural change in a highway viaduct. Through this network, we classified two
215 categories of signals: Damaged or healthy (or repaired) bridge. In both cases, the bridge's
216 natural frequency, detectable by oscillations due to the passage of vehicles, undergoes a
217 shift that is typically difficult to identify in the presence of noise caused by environmental
218 stimuli and variable traffic conditions. A spike neuron model is considerably more
219 complex than an artificial neuron model (accumulation and threshold), so its training
220 and inference require higher computational effort than its simplified version. To train
221 the LSNN model, we applied Backpropagation Through Time (BPTT) algorithm [19].

222 The network used in this work is described in [19]. The input layer is composed of
223 input neurons (I) which are connected in an all-to-all fashion through the W^I matrix to
224 the recurrent layer, which is composed of Adaptive-Integrate and Fire neuron (ALIF).
225 The recurrent layer is connected recursively to itself with an all-to-all connection matrix
226 W^H , and it is linked to the output layer in an all-to-all fashion with the matrix W^O .
227 The ALIF neurons are described by two state variables v and a . The first one is called
228 membrane potential and increase when the neuron receives a stimulus (spike or current).
229 When the v reach a value called v_{th} , it emits a spike. The ALIF neurons have a changeable
230 v_{th} ; this behaviour is described by a the second state variable. The following equations
231 describe an ALIF neuron:

$$232 \quad v_j^t = e^{-\frac{\delta t}{\tau_m}} v_j^{t-1} + \underbrace{\sum_{i \neq j} W_{ji}^H z_i^{t-1}}_{\text{ALif} \rightarrow \text{ALif}} + \underbrace{\sum_n W_{jn}^I x_n^t}_{\text{In} \rightarrow \text{ALif}} - \underbrace{v_{th} z_j^{t-1}}_{\text{Reset}} \quad (1)$$

$$233 \quad a_j^t = e^{-\frac{\delta t}{\tau_a}} a_j^{t-1} + z_j^{t-1} \quad (2)$$

$$234 \quad A_j^t = v_{th} + \rho a_j^t$$

$$235 \quad z_j^t = \begin{cases} 1 & \text{if } v_j^t \geq A_j^t \text{ and } r_j^t \neq 1 \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

232 Equation 1 describes the update of the membrane potential. α is the decay of the
233 neuron, and it depends on the tick (δt) of the network and the membrane time constant
234 τ_m . The second and the third term describe the contribution of the recurrent part and
235 the input layer, respectively. In the end, there is the reset of the membrane potential if
236 this reaches the v_{th} , z are the spikes of the ALIF, while x can be either spikes or current.
237 Equation 2 describes the update of the spike threshold of the ALIF, ζ is the decay of
238 the adaptive threshold, and it depends on δt and τ_a called decay time constant, a is
239 rescaled by a factor ρ before being added to v_{th} . Equation 3 describe the spike condition.
240 The neuron can spike (fire) only if it reaches a certain value (A) and if it is not in the
241 refractory period (r). The refractory period is triggered when a neuron spike, and it is a
242 time-lapse in which the neuron cannot fire.

The output neurons are continuous; therefore, the output is not a spike train but a
continuous waveform. The following equation describes the outputs neurons:

$$243 \quad y_k^t = e^{-\frac{\delta t}{\tau_o}} y_k^{t-1} + \underbrace{\sum_j W_{kj}^O z_j^t}_{\text{ALif} \rightarrow \text{Out}} + \underbrace{b_k}_{\text{Bias}} \quad (4)$$

243 where τ_o is the decay constant of the membrane potential of the neurons and b is the bias
244 of the neuron, which represents a constant current that stimulates the neuron.

245 2.4. Input Encoding Methods

246 The LSNN we used in this work can work with two types of input encodings:
247 Current (current-driven) or events (events-driven). In the Current-Driven LSNN, the
248 input signal is constant for all the inference time, while in the Events-Driven LSNN, the
249 signal is first encoded as spikes and then provided as input. This section describes some
250 of the state-of-the-art encoding methods and details the algorithm we implemented to
251 encode acceleration signals coming from the MEMS sensor. Considering Figure 2, which
252 shows the frequency shift that we want to detect, we give as input to the network the
253 FFT of the acceleration signal due to the vehicle crossing the viaduct.

254 Literature is rich in algorithms to encode a waveform into a stream of spikes. Some
255 of those approaches try to minimise signal reconstruction error, while others focus on
256 emulating biological-plausible behaviours.

257 The authors of [30] propose a family of methods that minimise signal reconstruction
258 error. All proposed methodologies are characterised by the presence of two complemen-
259 tary neurons (normally referred to as *positive* and *negative*), which expose a contrasting
260 behaviour, that is, when one of the two fires, the other does not.

261 The simplest temporal encoding algorithm is called *Threshold Based Representation*
262 (TBR). In this algorithm, whenever the difference between two consecutive signal sam-
263 ples is higher than a predetermined fixed threshold, then the *positive* neuron emits a
264 spike. Unfortunately, while being computationally cheap to implement, TBR is known
265 for leading to high reconstruction error [30], even for signals with simple dynamics.

266 The *Step Forward* (SF) method uses a baseline value (initialised as the value of the
267 first signal sample) and a fixed threshold. Suppose the absolute value of the difference
268 between two consecutive samples is higher than the sum of baseline and threshold. In
269 that case, the *positive* neuron spikes and the baseline is updated, adding the baseline. A
270 variation of this encoding strategy is called *Moving Window* (MW), where the baseline is
271 updated looking at a moving window of signal samples. The other encoding methods
272 proposed in [30] have not been considered in this work because of their inherent higher
273 computational complexity that is not suitable for the chosen architectural target.

274 In [31], the authors describe some biological methods of encoding without consid-
275 ering the reconstruction error. In all these methods, the information is encoded in the
276 reciprocal spikes of several neurons, meaning that proper encoding of the signal depends
277 on the number of neurons adopted. At the time of the first spike, the information is
278 stored in the delay between the start of the stimulus and the neuron's firing. In this
279 method, the first neuron inhibits all the others; therefore, the information is in just
280 one spike. In latency code, the information lies in the time between spikes of different
281 neurons. In Rank-Order Coding (ROC), the information is encoded in the order of the
282 spikes. In this method, every neuron can fire at most once for every sample (representing
283 a single FFT in our case).

284 3. Brain Inspired Processing

285 This section describes the two main components of the brain-inspired processing
286 pipeline, namely input encoding and LSNN architecture. Next section will describe the
287 optimization performed to improve LSNN implementation for edge devices.

288 3.1. LSNN Input Coding

289 In the previous section, we discussed possible encoding methods proposed in the
290 literature for LSNNs. In this section, we describe the method we applied to SHM data
291 in our brain-inspired processing pipeline. An example of the input signal is shown in
292 Figure 2.

293 In Figure 4 the entire data flow is shown. After collecting the data from the sensor,
294 we extract only the z-axis values since they are more sensitive to the vehicles passages.
295 These data are collected with a sampling frequency of 100 Hz and successively down-

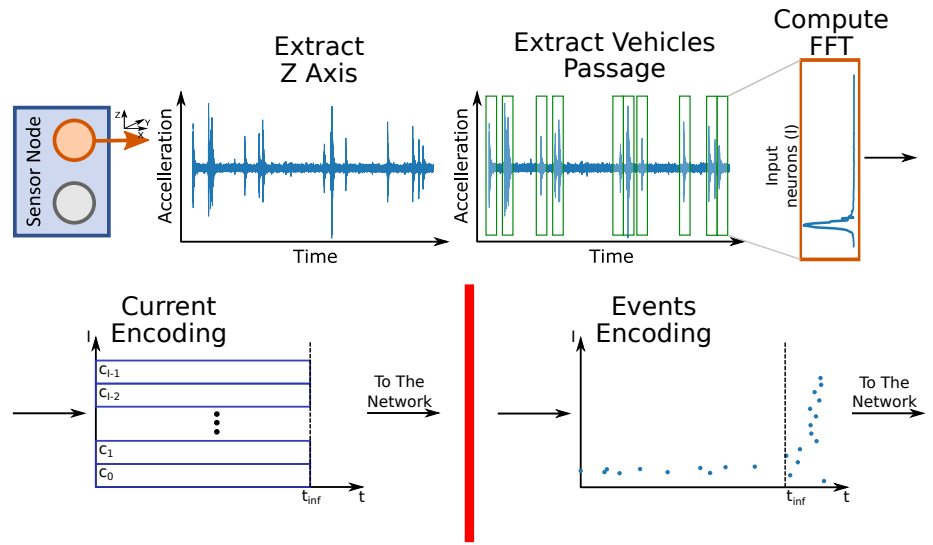


Figure 4. Data preprocessing.

296 sampled to 12.5 Hz. Subsequently, we get only the windows where vehicles passages
 297 have been detected. This was done by thresholding the signal.

298 The FFT of these windows is computed. The number of input neurons has been
 299 selected as two possible values: 50 or 150. This choice was made because a larger number
 300 of inputs would have caused a matrix of input weights (W^I) too large to train for such a
 301 small network, while a smaller number of inputs risks not extracting enough features
 302 from the signal [32]. So each vehicle's window can be composed of 100 or 300 coefficients
 303 that, in time, is equal to 8 or 24 seconds.

304 FFT coefficients are then given as input to the encoding stage. In the case of Current-
 305 Driven encoding, they are provided to the LSNN network one to each neuron for a
 306 constant time corresponding to t_{inf} . On the other side, in the case of Event-Driven
 307 encoding, the ROC algorithm is applied.

308 Each coefficient is thus encoded as a spike time interval. The larger the coefficient,
 309 the smaller the "time-to-spike". As a result, the spike time interval is inversely pro-
 310 portional to the value of the coefficients. As such, higher coefficients, which are more
 311 relevant for the damage detection because they have more energy, will fire first. Lower
 312 coefficients will fire later. Since t_{inf} is the inference time for each sample, all neurons that
 313 have not fired for $t < t_{inf}$ will no longer be able to fire. This implies that the coefficients
 314 with low energy (no information) will not impact the network input. The applied ROC
 315 encoding algorithm is shown below:

Algorithm 1: Compute Time-to-Event

input : A signal S of I coefficients and a inference time t_{inf}
output: A Time-to-Event T array of I spiking intervals
 $m = \min(S)$;
 $M = \max(S)$;
for $i \leftarrow 0$ **to** $I - 1$ **do**
 $t = (S[i] - m) / (M - m)$;
 $t_{check} = \text{round}(1/t)$;
 if $t_{check} \leq t_{inf}$ **then**
 | $T[i] = t_{check}$
 end
end
return T ;

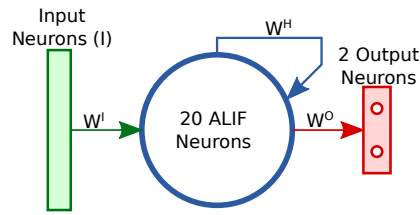


Figure 5. SNN architecture.

Parameter	Description	Explored Values
I	Input neurons	50, 150
τ_m	Recurrent layer membrane potential decay	20, 30
τ_o	Output layer membrane potential decay	3, 10, 30
v_{thc}	Spike threshold coefficient	0.01, 0.03
β_c	Adaptive threshold coefficient	1.7, 1.8
τ_{ac}	Adaptive threshold decay	0.5, 1.0
t_{inf}	Inference ticks	5, 10, 20
reg_c	Loss regularization coefficient	1, 100, 300
reg_r	Firing rate regularization coefficient	0.01, 0.001

Table 1: LSNN parameters explored

316 The signal S is the sequence of FFT coefficients corresponding to the observed
 317 acceleration window made of I coefficients. The inference time t_{inf} is also an input of
 318 the algorithm as it is a network parameter.

319 3.2. LSNN Architecture

320 The proposed LSNN network is shown in Figure 5, and it is composed of 20 ALIF
 321 and two output neurons which discriminate between the two classes of the bridge. We
 322 isolated the most important configuration parameters of the training algorithms (hyper-
 323 parameters) and studied their impact in terms of LSNN computational complexity and
 324 internal activity with an exhaustive search. The hyperparameters that we explored
 325 can be divided into two classes, the network parameters and the training parameters
 326 (summarised in Table 1). The first class comprises all the constants that strictly concern
 327 the network, while the second class is composed of all the values used to reach better
 328 performances but that don't feature the network. The network parameters that we
 329 explored are:

- 330 • The input number (I) [50, 150] is the number of input neurons in the first layer. A
 331 high number create a huge W^I matrix that cannot be trained by a small network,
 332 while a low number risks not extracting all the features of the input signal [32].
- 333 • τ_m [20, 30] represents the decay of the membrane potential in the recurrent layer.
 334 With high values, the neuron will need more time to return to the resting potential,
 335 representing the membrane potential of the neuron if any inputs have not perturbed
 336 it.
- 337 • τ_o [3, 10, 30] is the decay of the membrane potential of the output neurons.
- 338 • v_{thc} [0.01, 0.03]. The spike threshold (v_{th}) of the neurons is computing as: $v_{th} =$
 339 $v_{thc} / (1 - e^{-\delta t / \tau_m})$. A high value of the v_{thc} is translated into a higher value of the
 340 threshold.
- 341 • β_c [1.7, 1.8] is used to compute the increase of the adaptive threshold (ζ) in the ALIF
 342 neurons. Higher values of this parameter are equivalent to a greater increase in the
 343 spike threshold.
- 344 • τ_{ac} [0.5, 1] is used to compute the decay of the adaptive threshold (τ_a). τ_a should
 345 have a value comparable to the time length of the problem [19].

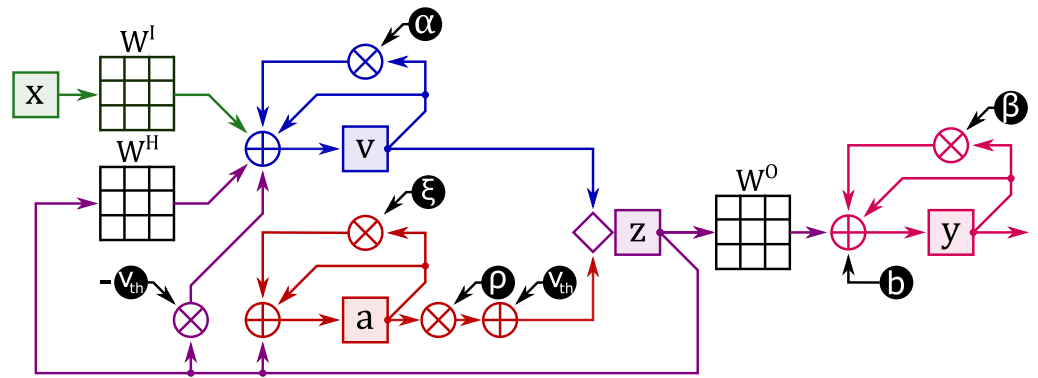


Figure 6. Flowchart of the LSNN update cycle. Neuron variables are represented with rectangles. Simple mathematical operations are represented with circles. Matrix multiplication operations with synaptic weights are represented by grids. Constants are represented with black circles. Data movements are represented by arrows between operators and operands.

346 The training parameters that we explored are the inference time (t_{inf}) expressed in ticks
 347 [5, 10, 20]. A tick in an SNN represents a cycle to complete the dynamics integration of
 348 all network neurons. The actual time depends on the implementation. The inference
 349 time is then the number of ticks that the network uses to process the input sample. For
 350 the regularisation coefficient we use (reg_c) [1, 100, 300] and for the regularisation rate we
 351 use (reg_r) [0.01, 0.001]. The two regularisation values are used to computing the loss of
 352 the network:

$$L_{fr} = \frac{1}{2} \sum_j \left(\frac{1}{nT} \sum_{t=1}^{nT} z_j^t - reg_r \right)^2 \quad (5)$$

$$L = L_p + L_{fr} * reg_c \quad (6)$$

353 Loss is composed of two parts (eq. 6), the loss of the classification (L_p) that is computing
 354 with the cross-entropy and the weighted firing-rate loss (L_{fr}). The L_{fr} is computed as
 355 the difference between the firing rate activity of the network and the reg_r that is the
 356 desired firing rate [19].

357 4. eLSNN: The Optimized Embedded LSNN

358 The implementation of LSNNs was performed taking into account resource resource
 359 constraints and exploiting SNN properties. In particular, sparsity in neuron response (e.g.
 360 firing) was exploited in order to skip processing cycles. The firing activity of neurons can
 361 be tuned as described in [19] using a loss value dedicated to limit the neuron activity.

362 Our LSNN implementation is depicted in Figure 6 and consists of four main steps:
 363 (i) Membrane voltage update (v), (ii) Membrane voltage threshold update (a), (iii)
 364 Evaluation of the spike emission (z), (iv) Evaluation of the output (y). The diagram
 365 shows: x data dependencies in green, v data dependencies in blue, a data dependencies
 366 in red, z data dependencies in purple and y data dependencies in magenta. In the
 367 next sections we will describe in detail the network implementation and the performed
 368 optimisations.

369 4.1. Membrane voltage update

370 The computation involved in the membrane voltage update (v), as shown in Figure
 371 6 in the first blue \oplus operator, is done through the following steps:

- 372 1. Membrane voltage decay: $v^{(\alpha)} := \alpha v$
- 373 2. Contribution of inputs on membrane voltage: $v^{(I)} := W^I x$
- 374 3. Contribution of internal neuronal activity on membrane voltage: $v^{(H)} := W^H z$
- 375 4. Reduction of membrane voltage in case of previous spike emission: $v^{(th)} := -v_{th} z$

Item	Naive			Optimised		
	sum	mul	store	sum	mul	store
$v^{(a)}$	-	1	-	-	1	-
$v^{(I)}$	I-1	I	-	$\tilde{x} - 1$	-	-
$v^{(H)}$	N-1	N	-	$\tilde{z} - 1$	-	-
$v^{(th)}$	-	1	-	1	-	-
v	4	-	1	4	-	1
Total	I+N+2	I+N+2	1	$\tilde{x} + \tilde{z} + 3$	1	1

Table 2: Operations analysis for a single neuron. These operations must be executed for each neuron (N) and for each inference tick (t_{inf}).

376 5. Update of membrane voltage: $v \leftarrow v + v^{(a)} + v^{(I)} + v^{(H)} + v^{(th)}$

377 In Table 2 we report the operations (sum and multiply) of both the non optimized
378 (Naive) and optimised version for each component of v . Overall, the operations to be
379 performed are: $(I + N + 2)Nt_{inf}$ sum and $(I + N + 2)Nt_{inf}$ multiplication.

380 Because of the event-driven design of the network, it is possible to implement the
381 computation of the $v^{(I)}$ and $v^{(H)}$ components more efficiently. Considering the row-
382 column multiplication between the matrix W (with $N \times M$ elements) and the column
383 vector x (with $M \times 1$ elements), the operation result will be added to the vector v . When
384 the elements of x can only assume binary values ($x_i \in \{0, 1\}$) the Wx operation can
385 be implemented using only sums. The following pseudo-code formally describes the
386 operation:

Algorithm 2: Optimized implementation of $v \leftarrow Wx$ when x is a boolean vector.

```

i ← 0;
while i < events do
  offset ←  $\tilde{x}[i]$ ;
  while n ≤ neurons do
    v[n] ← v[n] + W[offset];
    offset ← offset + neurons;
  end
  i ← i+1;
end

```

387 Let \tilde{x} be a list containing only the index of the non-zero elements of x . By iterating
388 over the elements of \tilde{x} we select the columns of W to be considered in the sum cycle (inner
389 cycle). The sum cycle iterates over the rows of the selected column, and for each element
390 accumulates its content in the result vector. The required operations will therefore be
391 dependent on the number of non-zero elements in the vector x . The number of non-zero
392 elements in the vector x will change with each inference tick. We then identify using the
393 symbol \tilde{x} the average number of non-zero elements of x for each inference tick.

394 We then reduce the sums required for the membrane voltage update from $(I +$
395 $N + 2)Nt_{inf}$ to $(\tilde{x} + \tilde{z} + 3)Nt_{inf}$ and the multiplications from $(I + N + 2)Nt_{inf}$ to Nt_{inf} .
396 The value of \tilde{x} depends on the chosen event encoding. The value of \tilde{z} depends on the
397 behaviour of the network. When training the network, it is possible to minimise \tilde{z} by
398 introducing its value into the loss calculation.

399 4.2. Threshold update

400 Updating the threshold for the membrane voltage (a), as shown in Figure 6 in the
401 first red \oplus operator, is broken down into the following operations:

- 402 1. Evaluation of threshold decay: $\mathbf{a}^{(\xi)} := \xi \mathbf{a}$
 - 403 2. Threshold adjustment in case of previous spike: $\mathbf{a}^{(z)} := z$
 - 404 3. Threshold update: $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{a}^{(\xi)} + \mathbf{a}^{(z)}$
 - 405 4. Weighted threshold computation: $a^{(\rho)} := \rho a$
 - 406 5. Reference threshold augmentation $v_{th}^{(a)} := v_{th} + a^{(\rho)}$
- 407 In total, $3Nt_{inf}$ sums and $2Nt_{inf}$ multiplications are required to implement the adaptive
408 threshold functionality.

409 4.3. Spike emission check

410 At each inference tick, the spike firing condition must be checked. For each neuron,
411 the membrane voltage is compared with the spike threshold. In this particular imple-
412 mentation, the threshold voltage is adapted to the activity of the neuron and increases as
413 its activity increases. A spiking neuron is also inhibited for a period t_{ref} called refractory
414 time. Even if the neuron has a membrane voltage above the threshold during this period,
415 it will not fire. To check this condition, each neuron stores the information about the tick
416 of the last spike $t_i^{(z)}$.

417 When checking the emission of a spike, as shown in Figure 6 in the first purple \diamond
418 operator, two conditions must therefore be checked for each neuron:

- 419 1. The membrane voltage must be above the adaptive threshold: $v_i \geq v_{th}^{(a)}$
- 420 2. The neuron must not be inhibited by an earlier spike: $t \geq t_i^{(z)} + t_{ref}$

421 If the above conditions are satisfied, the vector \mathbf{z} will take the value 1 at position i ,
422 otherwise the value 0.

423 In our implementation, instead of directly handling the vector \mathbf{z} , we use the list of
424 events $\check{\mathbf{z}}$. Using the list of events, we can replace matrix multiplications $W^H \mathbf{z}$ and $W^O \mathbf{z}$
425 with sums. At the beginning of the spike emission check phase, the list $\check{\mathbf{z}}$ of previous
426 events are cleared. In the presence of a spike emission, the identifier of the spiking
427 neuron will be added into the $\check{\mathbf{z}}$ list.

428 4.4. Output update

429 Output neurons receive spikes generated by network neurons in the recurrent layer
430 at the same tick as they are emitted. As shown in Figure 6 in the first magenta \oplus operator,
431 the output neurons have a similar update cycle to the recurrent neurons:

- 432 1. Output decay: $\mathbf{y}^{(\beta)} := \beta \mathbf{y}$
- 433 2. Contribution of internal neuronal activity on output: $\mathbf{y}^{(H)} := W^O \mathbf{z}$
- 434 3. Bias contribution: $\mathbf{y}^{(b)} := \mathbf{b}$
- 435 4. Update of output: $\mathbf{y} \leftarrow \mathbf{y} + \mathbf{y}^{(\beta)} + \mathbf{y}^{(H)} + \mathbf{y}^{(b)}$

436 Again, the calculation of $\mathbf{y}^{(H)}$ requires a multiplication between the matrix W^O and
437 the vector \mathbf{z} . The procedure described in 4.1 helps to lower the number of operations.
438 Using the list $\check{\mathbf{z}}$ it is possible to solve the operation $W^O \mathbf{z}$ using sums only. The number
439 of operations is then lowered from $(N - 1)Ot_{inf}$ sums and NOt_{inf} multiplications to
440 $\check{z}Ot_{inf}$ sums.

441 4.5. Current-driven input

442 In this work, we consider also an alternative to input events, using continuous
443 (e.g. real) values in the vector \mathbf{x} [19]. While in this case it is not possible to perform
444 the optimisations described in Algorithm.2 for solving $\mathbf{v}^{(l)} := W^l \mathbf{x}$, we provide an
445 optimised version of the LSNN using this type of input to evaluate the trade-offs lead
446 by the two encoding methods and reported in Section 5. We note that in this case the
447 contribution of internal neuronal activity $\mathbf{v}^{(H)}$ (hidden/recursive layer) is still spike-
448 based, however in the input-layer it is not possible to remove multiply operations as it is
449 possible using input spike events.

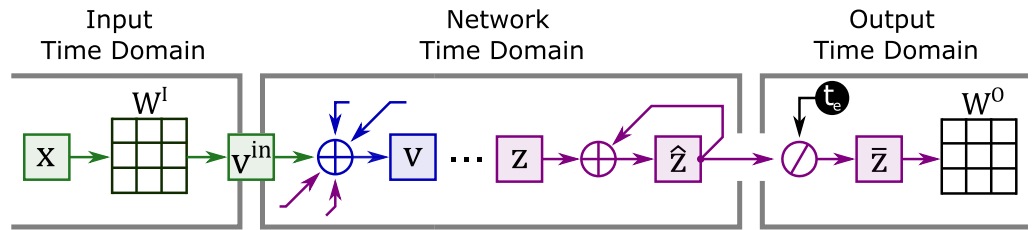


Figure 7. Flowchart modified to handle current-driven input. In this application scenario the input and output have two different time domains from the network. v^{in} and \hat{z} vectors are used to pass information from the input to the network and from the network to the output.

450 In the literature, networks using continuous input typically observe a sample for a
 451 period called the exposure time t_{exp} . Within the network exposure time, the same values
 452 of x are always presented. This leads to the formation of three time domains (although it
 453 is common for the time domains of input and output to coincide) represented in figure 7:

- 454 • Temporal domain of the input, where the input varies over time.
- 455 • Temporal domain of the network, for each tick in the input-time-domain the net-
 456 work performs t_{exp} iterations always observing the same value of x
- 457 • Temporal domain of output, where output varies over time.

458 In order to efficiently handle this LSNN variant, the time domains are decoupled by
 459 vectors v^{in} and \hat{z} . The first vector decouples the input from the network; it is computed
 460 by a row-column multiplication between the matrix W^l and the vector x each time the
 461 input changes. Then, the vector v^{in} will be used to increment v at each tick of the network.
 462 The number of operations to compute $v^{(l)}$ then becomes $(1/t_{exp}(I - 1)N + N)t_{inf}$ sums
 463 and $t_{inf}/t_{exp}IN$ multiplications.

464 The second vector decouples the network from the output. The output will no
 465 longer process the spikes coming from the network but will use the average activity of
 466 the network within the exposure time as information. At each tick, the network must
 467 accumulate the emitted spikes inside the vector \hat{z} . In the time domain of the output the
 468 content of the vector will be averaged (\bar{z}) and used for the calculation of $y^{(H)} := W^o\bar{z}$.

469 The number of operations for calculating $y^{(H)}$ then becomes $(1/t_{exp}(N - 1)O +$
 470 $\bar{z})t_{inf}$ sums and $t_{inf}/t_{exp}NO$ multiplications.

471 In the following, we will refer to this LSNN version as *current-driven* eLSNN, since
 472 the continuous and constant input over the exposure time can be interpreted as a constant
 473 current stimulus. The version working with input spikes (e.g. binary values) will be
 474 referred to as *event-driven* eLSNN instead.

475 5. Experimental Result

476 This section first describes the experimental setup used to test the LSNN imple-
 477 mentation presented in the previous sections. Then, it reports the results of the design
 478 trade-off characterisation for the proposed eLSNN. In particular:

- 479 • Section 5.1 describes the Hardware-in-the-Loop system we implemented to profile
 480 the runtime and energy performance of the SHM application.
- 481 • In Section 5.2, we study the accuracy of the trained eLSNNs (current-driven and
 482 event-driven). We evaluated the MCC on the test dataset for the first order hyper-
 483 parameters, which impact the energy and computational efficiency of the eLSNN
 484 implementation. The first-order hyper-parameters are the input number (I) and
 485 inference ticks (t_{inf}) of the networks.
- 486 • In Section 5.3, we study the impact of the eLSNN performance ($\#execution\ cycles$)
 487 considering the most accurate networks for each eLSNN version (current-driven
 488 and event-driven). We also considered different combinations of the first-order
 489 parameters as a function of the activity factors of the eLSNNs (number of non-zero
 490 elements (spikes), both in the input (x) and hidden/recurrent layer z).

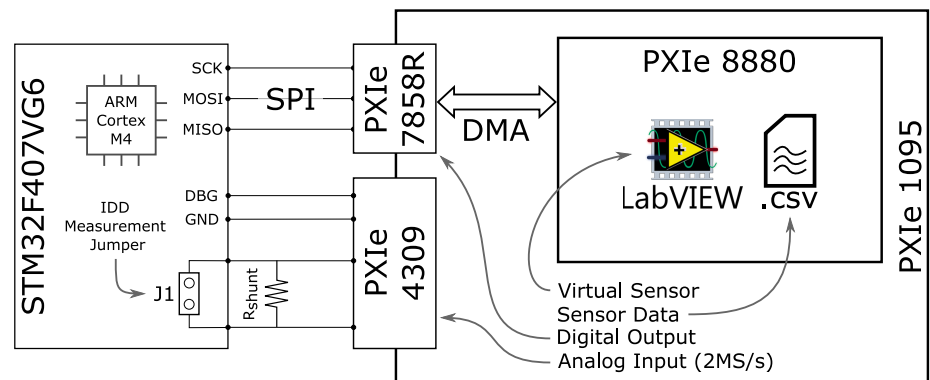


Figure 8. Architecture of the Hardware-in-the-Loop test system. The PXI system comprises a PXIe-4309 *Analog Input* module to monitor current consumption in each stage of the pipeline. The PXIe7858R FPGA, along with the PXIe-8880 Windows 7 controller, implements the *Virtual Sensor* and allow loading MEMS readings from a CSV file in real-time, emulating a real buffered digital MEMS node in hardware. The whole application runs on a STM32F4 development board featuring a Cortex-M4 ARM core.

- 491 • In Section 5.4, we perform a complete characterisation of the SHM sensing node
 492 firmware for a network implementation having median activity-factors. This cor-
 493 responds to a typical behaviour of the sensor node for a real SHM application in
 494 terms of execution time and energy-consumption.

495 5.1. Testbed

496 We prepared a Hardware-in-the-Loop setup to characterise the performance and
 497 energy of the data acquisition and processing board. As shown in Figure 8, the testbed
 498 we implemented is composed of three actors: i) A development board hosting the
 499 STM32 MCU (*System-on-Chip*) on the left; ii) A data acquisition system to monitor the
 500 consumption of the STM32 development board (*Analog Input*) on the right and iii) A
 501 FPGA emulation module able to feed the processing unit in the system-on-chip with
 502 data coming from the real-world SHM application described in Section 2 in real-time
 503 (*Virtual Sensor*). (based on PXIe systems from NI [33])

504 Both the *Analog Input* and the *Virtual Sensor* are part of a PXIe systems from NI [33]).
 505 This is a modular, LabVIEW controlled environment able to perform fast measurements
 506 requiring high-performance digital and analog I/O. The following three subsections
 507 provide further details about the PXI modules used in the experimental setup and their
 508 role in the pipeline characterisation.

509 5.1.1. System-on-Chip

510 We implemented and deployed the entire SHM pipeline on an STM32F407VG6
 511 MCU. It is a system-on-chip manufactured by STMicroelectronics, which features an
 512 ARM Cortex-M4 with a floating-point unit. The core comes with 192 KB RAM and
 513 1 MB FLASH memory and supports a maximum clock frequency of 168 MHz. The
 514 STM32F407VG6 supports multiple low-power modes and various clock frequency con-
 515 figurations, which allow a fine-grained tuning of the system performance depending on
 516 the application needs. For the sake of this work, we apply the following policies:

- 517 • When the system is in RUN mode, the system-on-chip is always clocked at the
 518 maximum allowed clock frequency, equal to 168 MHz.
 519 • When the system fetches data from the MEMS sensor over SPI, the transfer is
 520 performed via DMA with the core in SLEEP mode. Before entering SLEEP mode,
 521 the MCU core is clocked down to 16 MHz to minimize SLEEP current consumption.
 522 • When the system-on-chip is not working, it is put in STANDBY mode, switching
 523 off the voltage regulator to achieve the lowest consumption possible. Notice that

524 we can apply such an aggressive power-saving policy since we assume our SHM
525 application does not need to retain any state before one LSNN inference and the
526 next inference.

527 Considering the software stack employed to implement the SHM application, we in-
528 terfaced the on-board peripherals using the Hardware Abstraction Layer provided by
529 STMicroelectronics. Instead, the mathematical processing (e.g. FFT computation, FFT-to-
530 spike conversion) is implemented using the CMSIS-DSP library primitives. It is a set of
531 routines developed by ARM to deliver DSP-like functions optimized to run on Cortex
532 cores.

533 5.1.2. Analog Input

534 The power and performance monitoring activity of the pipeline has two require-
535 ments: i) Measure the current sunk by the system-on-chip at any time of the data
536 acquisition and processing and ii) precisely split the current waveform into each pipeline
537 stage, to detect the analysis stages that are the more power hungry or require most MCU
538 cycles to be accomplished. Both requirements are met by using a PXIe-4309 ADC device
539 reading 2M Samples per second and featuring 32 channels 18 bit wide. We used two of
540 the available input channels to perform the following synchronous activities:

- 541 • Sample the current sunk by the MCU using a 1 Ohm shunt resistor. Such measure-
542 ment exploits a jumper available on the development board to monitor current
543 consumption on the VDD of the system-on-chip.
- 544 • Read the logic level of a debug GPIO (DBG pin, Figure 8) that is toggled by the
545 application software each time a processing stage is started or completed, to asso-
546 ciate each phase (e.g. SPI transfer, FFT computation, SNN inference) to its current
547 consumption waveform (with negligible overhead on the application performance).

548 5.1.3. Virtual Sensor

549 The *Virtual Sensor* is an FPGA-based emulation system that allows feeding the SPI
550 interface of the MCU with data taken from a trace. The trace was obtained from a real
551 sensor deployment on the field. For characterisation and profiling purposes, the virtual
552 sensor was connected to the MCU replacing the MEMS present in the sensor node board.
553 The *Virtual Sensor* is completely implemented within the PXI system and is made of two
554 modules sharing data through a DMA-controlled hardware FIFO:

- 555 • PXIe-8880: The PXI system controller, which is a general-purpose CPU-based host
556 running Windows 7 and LabVIEW. It loads the accelerations measured on the
557 bridge from a CSV file and pushes them into a FIFO at the boundary of the FPGA
558 system.
- 559 • PXIe-7858R: The PXI FPGA module, which loads the measurements the DMA move
560 from the controller to its FIFO. It acts as a digital MEMS that samples structural
561 accelerations at a constant rate and stores them in an on-board buffer which can be
562 accessed in-order through an SPI interface.

563 5.2. Accuracy vs. first-order hyper-parameters

564 As introduced in Section 2, to study the impact of the eLSNNs hyper-parameters
565 and flavours on the accuracy, we conducted an exhaustive search. We evaluated the
566 accuracy as the Matthews Correlation Coefficient (MCC) on the test set, which includes
567 all the samples related to vehicle passages on the bride section during a randomly chosen
568 day before and after the scheduled maintenance intervention.

569 In Figure 9, we report on the y-axis the network accuracy measured as MCC. We
570 limited the plot to those hyper-parameters configurations leading to eLSNNs with an
571 $MCC \geq 0.6$. This corresponds to an accuracy of 0.77. Moreover, in the same figure, we
572 draw the line at the $MCC = 0.75$. We consider all the eLSNNs configurations achieving an
573 MCC higher than 0.75 as acceptable or "good" configurations. This threshold corresponds
574 to accuracy above 0.88.

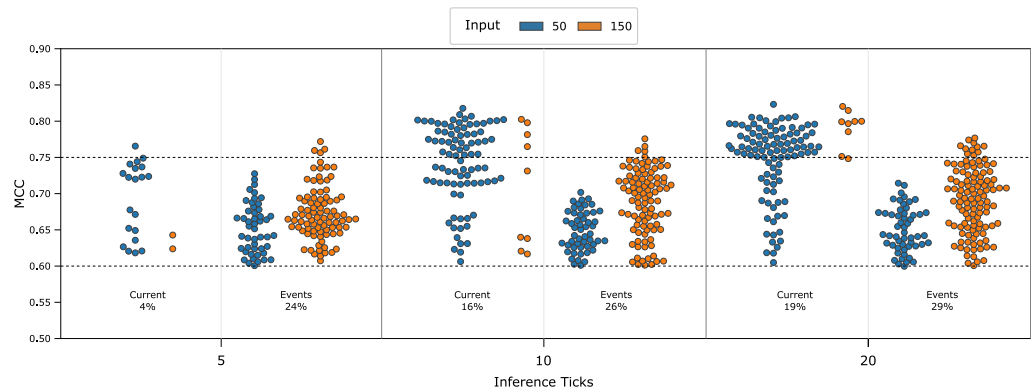


Figure 9. Inference ticks and number of inputs impact on MCC using the Current Input and the Event Input. Each point is a configuration of LSNN, which is characterised by its hyperparameters.

575 On the x-axis, we report three sets of plots. Each set corresponds to different
 576 inference ticks ($t_{inf} = 5, 10, 20$). Inside each set, the left plot refers to the current-driven
 577 eLSNN, and the right plot refers to the event-driven eLSNN. Inside each of these plots,
 578 we report the eLSNN configurations achieving a $MCC > 0.6$. The percentage of these
 579 configurations on the total evaluated (1728 for each eLSNN flavour) is reported in the
 580 text on each plot's bottom. The MCC accuracy of each configuration is reported with red
 581 bins for an input number of 50 and with blue bins for an input number of 150 bins. In
 582 the SHM application, the input number (I) corresponds to the magnitude of the spectral
 583 components of the FFT. It thus is equal to the double of the accelerations samples, which
 584 needs to be read by the sensor to compute an eLSNN inference.

585 The combination of these parameters (t_{inf} & I) constitutes the first-order hyper-
 586 parameter that, as we will see in the next section, impact the inference execution time
 587 of the eLSNNs and then their energy consumption. From the figure, we can notice
 588 that for the current-driven eLSNNs, the largest number of acceptable configurations is
 589 achieved for an input number (I) of 50. Which also corresponds to the lower complexity
 590 of the eLSNN computations (see Section 4). Differently, for the event-driven eLSNNs,
 591 acceptable configurations can be achieved only with an input number (I) of 150. As
 592 we will see in Section 5.4, this has a severe drawback on the pre-processing cost for this
 593 eLSNN flavour. It is also worth to note that for the current-driven eLSNNs the accuracy
 594 improves with larger inference ticks, having any acceptable configurations with $I = 50$
 595 and $t_{inf} = 5$ but several with $I = 50$ and $t_{inf} = 10, 20$. However, this is not the case
 596 for the event-driven eLSNNs for which their MCC does not improve significantly with
 597 progressive t_{inf} increases. It is now interesting to evaluate these parameters' impact in
 598 terms of eLSNN inference execution cycles.

599 5.3. Execution time vs. activity-factors

600 As described in Section 4, the optimised eLSNN algorithm we propose in this paper
 601 leverages the sparse nature of the spikes for saving computations. If a neuron does
 602 not receive a spike, it does not trigger accumulation in the membrane potential. This
 603 means that the computational burden of the eLSNN algorithm depends on the average
 604 number of non-zero elements for each inference tick in the recursive/hidden layers. This
 605 is true for both current-driven and event-driven eLSNNs. For the event-driven eLSNN
 606 only, also the input layer must be considered in this computation. To understand the
 607 relevance of these effects on the total eLSNN inference time, we have to analyse different
 608 input samples. Each sample corresponds to the FFT coefficients of the accelerations
 609 read by the MCU during a passage of a vehicle. More specifically, the number of spikes
 610 corresponding to non-zero elements in the recurrent layer during an eLSNN inference
 611 (denoted with z) depends on the specific input sample and network hyper-parameters.
 612 Differently from the recurrent/hidden layer activity (z), the input activity factors vary

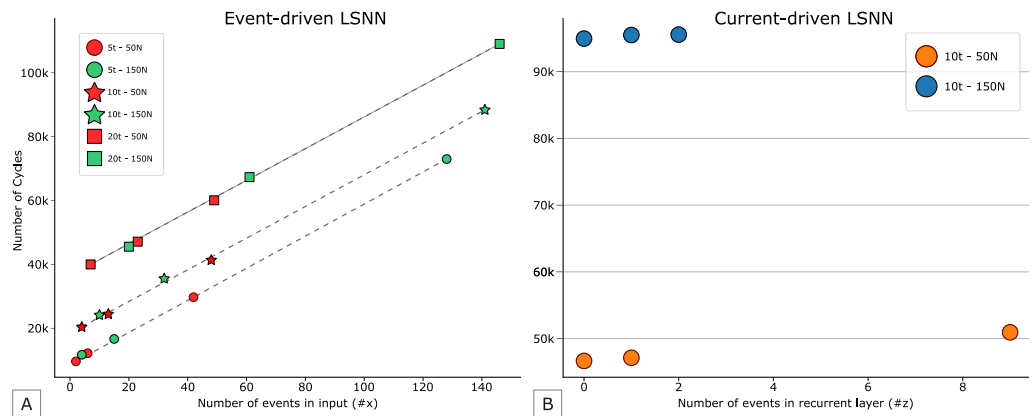


Figure 10. A) Cycle count for the event-driven eLSNN for the best configurations (input number with different colours, and inference ticks with different markers) computed on minimal, median and maximal spike activities on the input-layer (x). B) Cycle count for the current-driven eLSNN computed on minimal, median and maximal spike activities on the recurrent/hidden-layer (z). Different colour represent different number of inputs configuration for the eLSNN. All networks have an 10 inference ticks.

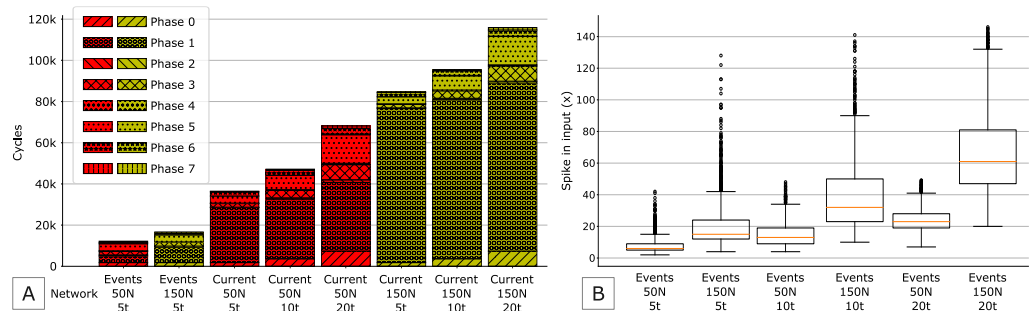


Figure 11. A) cycle count breakdown for the event-driven (event) and current-driven (current) eLSNN best configurations (input number - inference ticks) computed on median activity conditions. B) Distribution of the spikes in input for the event-driven eLSNN with different configurations (input number - inference ticks).

613 between the eLSNN flavour. For the current-driven eLSNNs, the input activity depends
 614 only on the input number (I). In contrast, for the event-driven eLSNNs, the input layer
 615 activity depends on the input spikes and non-zero elements in the coded inputs for all
 616 the ticks (x). x depends on the input sample and input number (I) as it is a property of
 617 the spiking input encoding.

618 Figure 10.B reports on the y-axis the total number of cycles needed by the eLSNN
 619 to complete one inference computation on a given input sample. This number of cycles
 620 accounts only for the eLSNN computation after the pre-processing step. On the x-axis,
 621 we report the value of z , which corresponds to the number of spikes/events/non-zero
 622 elements in the recurrent/hidden layer integrated into all the inference ticks (t_{inf}).

623 The different colours refer to the two different networks selected among the many
 624 with acceptable performance. For each type of network, we selected the ones with a
 625 number of inputs (I) leading to the largest (z) variation among the input samples in the
 626 test set.

627 Then, we plot three values of z chosen for each network corresponding to the
 628 minimum, the maximum, and the median sample. From the plot, we can notice that
 629 the impact of the z is negligible with respect to the total execution time of the eLSNN
 630 inference. Differently, the inference time for current-driven eLSNN halves when reducing
 631 the input number I from 150 to 50. As the impact of z is negligible, we can ignore its
 632 effect in the following plots.

633 Figure 11.B reports on the y-axis the distribution of the non-zero elements in the
 634 input layer for each input sample in the test set for event-driven eLSNNs. The distribu-
 635 tion depends on the input element and the encoding, which depends on the first-order
 636 hyper-parameters, namely the inference ticks (t_{inf}) and the number of inputs (I). In the
 637 y-axis, we report the different configurations of the first-order hyper-parameters. From
 638 the plot, we can see that the number of events/spikes/non-zero elements in the input
 639 layer (x) is more significant and has higher variability than the same quantity in the
 640 recurrent/hidden layer (Figure 10.B). Moreover, the number of input events/spikes/not-
 641 zero elements increases both in average and standard deviation with the increase of the
 642 input number and inference ticks.

643 Figure 10.A reports for each of the different configurations of the first-order hyper-
 644 parameters and for the input sample corresponding to the minimum, maximum and
 645 median values of x the execution time of the event-driven eLSNN. We can see that both
 646 the first-order hyper-parameters vary the execution time. The execution time increases
 647 linearly with the inference ticks and with x . It is interesting to notice that for the event-
 648 driven eLSNNs, the input number (I) does not increase the execution time directly
 649 as with the current-driven eLSNNs but indirectly. Indeed, a larger input number (I)
 650 increases the execution time proportionally to the increase of x . Which eLSNN flavour
 651 and configuration should thus be preferred?

652 5.4. Event-driven vs. current-driven

653 This section concludes the eLSNN study by comparing the performance of the most
 654 performing and energy-efficient current-driven, and event-driven eLSNN averaged on
 655 the test set. This is obtained by evaluating the candidate eLSNNs in the median sample
 656 with respect to both x and z .

657 Figure 11.A reports on the same plot the breakdown of the total number of cycles
 658 taken by the median inference time of both the current-driven and event-driven eLSNNs.
 659 For the current-driven networks, all the first-order hyper-parameters combinations are
 660 reported, while for the event-driven eLSNNs, we report only the configurations with
 661 inference ticks equal to 5, which corresponds to the most energy-efficient networks. By
 662 comparing the different networks, we can notice that the two most performing eLSNNs
 663 which achieves $MCC \geq 0.75$ are the configuration $I = 150, t_{inf} = 5$ for the event-driven
 664 eLSNN and $I = 50, t_{inf} = 5$ for the current-driven eLSNN. The event-driven eLSNN
 665 requires less than half of the cycles of the current-driven eLSNN. We can conclude that
 666 event-driven eLSNN is significantly more efficient ($> 50\%$) than the current-driven
 667 eLSNN. It must be noted that this conclusion does not account for the pre-processing of
 668 the input sample (affecting the input number) which is more significant for the selected
 669 event-driven eLSNN. It is interesting to notice that the event-driven eLSNN configured
 670 with $I = 50, t_{inf} = 5$ is the most efficient one, but its MCC is lower (0.72) than the
 671 accuracy threshold of $MCC \geq 0.75$. Moreover, in the Figure 11.A we report with different
 672 patterns the number of cycles needed to perform the computational steps described in 4.

- 673 • Phase 0: Compute of $v^{(\alpha)}$
- 674 • Phase 1: Compute of $v^{(I)}$
- 675 • Phase 2: Compute of $v^{(H)}$
- 676 • Phase 3: Compute of $a^{(\xi)}$
- 677 • Phase 4: Compute of $a^{(th)}$ and $a^{(z)}$
- 678 • Phase 5: Compute of z and insert items in \tilde{z}
- 679 • Phase 6: Compute of y
- 680 • Phase 7: Insert items in \tilde{x} for next iteration.

681 For the current-driven eLSNN the Phase 1 dominates the computational time since
 682 it must compute the v^{in} vector by means of a complete matrix-vector multiplication.
 683 Phase 0 and Phase 3 involves N multiplications, and at increasing t_{inf} their execution
 684 times become increasingly evident.

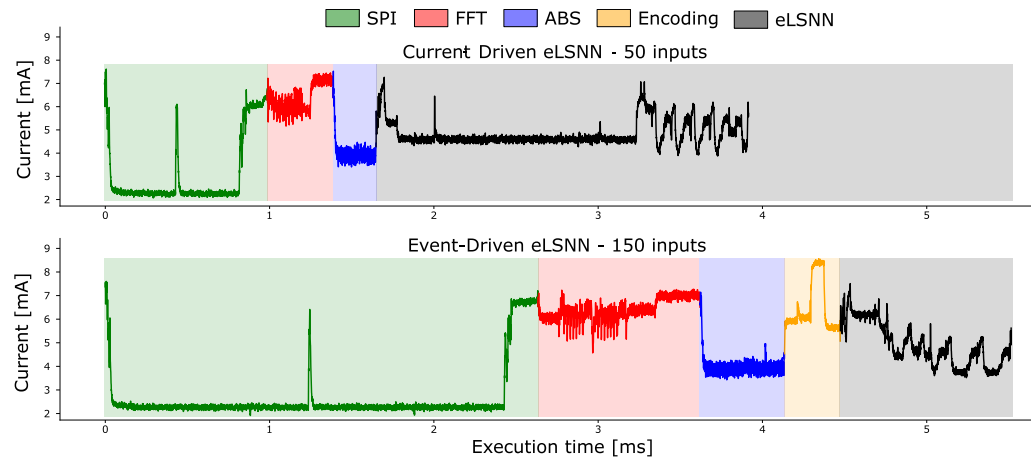


Figure 12. SHM application current consumption patterns for the best eLSNN networks working with current (top) and event (bottom) inputs. To better highlight the different stages of the application, the waveforms were obtained with the MCU clock slowed down to 16 MHz, while the SPI clock was 2 MHz.

Stage	Current-Driven				
	Time [μ s]	Cycles [#]	Current [mA]	Power [mW]	Energy [μ J]
SPI	542	-	5.01	16.53	8.96
FFT	41	6971	38.76	127.91	5.24
ABS	26	4284	36.58	120.71	3.14
Encoding	-	-	-	-	-
eLSNN	215	36036	40.26	132.86	28.56
Total	824	47291	-	-	45.90
Mean	-	-	16.88	55.70	-

Table 3: Power report of the SHM application featuring the best eLSNNs tested. The SPI master transfers data with a serial clock of 8 MHz. The SPI stage includes: i) The SPI data transfer, ii) the time required by the system-on-chip to reconfigure the clock to 168 MHz when it wakes-up from SLEEP mode and iii) the conversion of sensor data from raw integers to floating-point.

685 Figure 12 shows the MCU’s current consumption when executing the entire pro-
 686 cessing steps needed to read the sensor values through the SPI interface, pre-process
 687 the sample (FFT + ABS), and compute the eLSNN kernel. We report the current-driven
 688 eLSNN in its most efficient configuration on the left plot, and on the right plot, we report
 689 the event-driven eLSNN in its most efficient configuration. Even if the event-driven
 690 eLSNN kernel costs halves of the cycles than the current-driven eLSNN, it requires
 691 three times more sensor’s readings to compute an inference for an input sample. This
 692 increases the cycles needed to read the sensor’s data in SPI and compute the FFT and abs.
 693 Moreover, the event-driven eLSNN requires an additional pre-processing step consisting
 694 of the coding of the spectral components in spikes/events described in Section 3.

695 As described in section 4.5 the version of the network with current input needs
 696 a matrix-vector multiplication for each input tick $t_{inp} = t_{inf}/t_{exp}$. In this use case, we
 697 have only one input tick and only one matmul is executed, but it is enough to increase
 698 the number of cycles considerably. In Figure 12.A it is possible to appreciate this phase
 699 because it is visible for a long period (around milliseconds 2 and 3) not present in figure
 700 12.B.

701 Tables 3 and 4 summarises all these effects. We can notice that even if the event-
 702 driven eLSNN kernel takes 54% fewer cycles than the current-driven eLSNN network,
 703 the total computation time is 1.51x longer for the event-driven eLSNN. This is primarily

Stage	Event-Driven				
	Time [μ s]	Cycles [#]	Current [mA]	Power [mW]	Energy [μ J]
SPI	949	-	4.33	14.29	13.56
FFT	101	16968	38.17	125.96	12.72
ABS	50	8316	35.97	118.70	5.94
Encoding	32	5459	42.65	140.75	4.50
eLSNN	99	16631	38.20	126.06	12.48
Total	1231	47374	-	-	49.20
Mean	-	-	12.11	39.97	-

Table 4: Power report of the SHM application featuring the best eLSNNs tested. The SPI master transfers data with a serial clock of 8 MHz. The SPI stage includes: i) The SPI data transfer, ii) the time required by the system-on-chip to reconfigure the clock to 168 MHz when it wakes-up from SLEEP mode and iii) the conversion of sensor data from raw integers to floating-point.

704 due to the SPI transfer, which is 2x longer than for the current-driven eLSNN. Even with
705 this extra execution time, the total time for computing the network matches the real-time
706 requirements of the SHM application. Due to the lower power cost of the SPI transfer (w.
707 DMA), however, the energy-consumption for the eLSNN flavours (event-driven and
708 current-driven inputs) is comparable and in the range of 46 - 49 μ J.

709 In future works, we will explore temporal coding techniques and migrate the
710 coding task directly in the MEMS sensor and in the time domain. These on-sensor
711 coding techniques will remove the FFT cost and the amount of data to be read from the
712 MEMS sensors, achieving further energy reduction in the sensor node.

713 6. Conclusion

714 In this paper, we presented an optimised implementation of a recurrent Spiking
715 Neural Network on embedded microcontrollers for damage detection in Structural
716 Health Monitoring applications. We studied the feasibility of spiking based processing
717 and the trade-offs involved in using an event-based input. Thanks to this implementation,
718 called eLSNN, we were able to design, implement and characterise an SNN-based
719 monitoring system, where the computation is performed near to the sensor node. We
720 described the optimisations we performed with respect to a naive LSNN algorithm
721 present in the state-of-art to reduce computation cycles and improve energy efficiency.
722 We also studied two alternative encodings of the input, showing how they impact
723 performance and energy. We highlight the trade-offs between eLSNN execution and
724 data transfer costs that have to be explored to select the best energy/performance
725 configurations. Results of accuracy obtained on a real use case demonstrated that LSNN
726 is a viable solution for damage detection in SHM, having high accuracy ($MCC \geq 0.75$)
727 and low cycle and energy overheads if compared with the data transfer costs. Also,
728 the results highlight that moving the spike-coding directly in the sensor can lead to
729 even more energy-efficient implementation. The eLSNN working with input spikes is
730 more energy-efficient than the one using real (continuous) values. The promising results
731 in this work open the way to the implementation of SNN-based processing on edge.
732 Moreover, when compared with a state-of-art work leveraging an autoencoder (ANN)
733 on the same dataset [7], the proposed eLSNN approach shows lower complexity (from
734 32 000 MAC operations to 15 750 sums and 750 multiplications).

735 Future work will be devoted to evaluate and compare event-based SNN approaches
736 against other SHM datasets and comparing extensively with ANN algorithms.

737 **Acknowledgment**

738 This work was supported by INSIST Project (PON Ricerca e Innovazione 2014-
739 2020).

References

1. Smarsly, K.; Dragos, K.; Wiggenbrock, J. Machine learning techniques for structural health monitoring. Proceedings of the 8th European workshop on structural health monitoring (EWSHM 2016), Bilbao, Spain, 2016, pp. 5–8.
2. Burrello, A.; Marchioni, A.; Brunelli, D.; others. Embedded Streaming Principal Components Analysis for Network Load Reduction in Structural Health Monitoring. *IEEE Internet of Things Journal* **2021**, *8*, 4433–4447. doi:10.1109/jiot.2020.3027102.
3. Riziotis, C.; Testoni, N.; Aguzzi, C.; others. A Sensor Network with Embedded Data Processing and Data-to-Cloud Capabilities for Vibration-Based Real-Time SHM. *Journal of Sensors* **2018**, *2018*, 2107679. doi:10.1155/2018/2107679.
4. Foundation, T. TinyML. <https://www.tinyml.org>, 2021.
5. Microelectronics, S. STM32 solutions for artificial neural networks. https://www.st.com/content/st_com/en/ecosystems/stm32-ann.html, 2021.
6. Meyer, M.; Farei-Campagna, T.; Pasztor, A.; others. Event-Triggered Natural Hazard Monitoring with Convolutional Neural Networks on the Edge. Proceedings of the 18th International Conference on Information Processing in Sensor Networks; Association for Computing Machinery: New York, NY, USA, 2019; Ipsn '19, p. 73–84. doi:10.1145/3302506.3310390.
7. Moallemi, A.; Burrello, A.; Brunelli, D.; others. Model-based vs. Data-driven Approaches for Anomaly Detection in Structural Health Monitoring: a Case Study. 2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC). Ieee, 2021, pp. 1–6.
8. Maass, W. Networks of spiking neurons: The third generation of neural network models. *Neural Networks* **1997**, *10*, 1659–1671. doi:https://doi.org/10.1016/S0893-6080(97)00011-7.
9. Indiveri, G.; Horiuchi, T. Frontiers in Neuromorphic Engineering. *Frontiers in Neuroscience* **2011**, *5*, 118. doi:10.3389/fnins.2011.00118.
10. Roy, K.; Jaiswal, A.; Panda, P. Towards spike-based machine intelligence with neuromorphic computing. *Nature* **2019**, *575*, 607–617. doi:10.1038/s41586-019-1677-2.
11. Deng, L.; Wu, Y.; Hu, X.; others. Rethinking the performance comparison between SNNS and ANNS. *Neural Networks* **2020**, *121*, 294–307. doi:https://doi.org/10.1016/j.neunet.2019.09.005.
12. Lamba, S.; Lamba, R. Spiking Neural Networks Vs Convolutional Neural Networks for Supervised Learning. 2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), 2019, pp. 15–19.
13. Neves, A.C.; González, I.; Leander, J.; others. Structural health monitoring of bridges: a model-free ANN-based approach to damage detection. *Journal of Civil Structural Health Monitoring* **2017**, *7*, 689–702. doi:10.1007/s13349-017-0252-5.
14. Seventekidis, P.; Giagopoulos, D.; Arailopoulos, A.; others. Structural Health Monitoring using deep learning with optimal finite element model generated data. *Mechanical Systems and Signal Processing* **2020**, *145*.
15. Avci, O.; Abdeljaber, O.; Kiranyaz, S.; Inman, D. Structural damage detection in real time: implementation of 1D convolutional neural networks for SHM applications. In *Structural Health Monitoring & Damage Detection, Volume 7*; Springer, 2017; pp. 49–54.
16. Yang, L.; Fu, C.; Li, Y.; others. Survey and study on intelligent monitoring and health management for large civil structure. *International Journal of Intelligent Robotics and Applications* **2019**, *3*, 239–254. doi:10.1007/s41315-019-00079-2.
17. Pang, L.; Liu, J.; Harkin, J.; others. Case Study, Spiking Neural Network Hardware System for Structural Health Monitoring. *Sensors* **2020**, *20*. doi:10.3390/s20185126.
18. Madrenas, J.; Zapata, M.; Fernández, D.; others. Towards Efficient and Adaptive Cyber Physical Spiking Neural Integrated Systems. 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2020, pp. 1–4. doi:10.1109/icecs49266.2020.9294982.
19. Bellec, G.; Scherr, F.; Subramoney, A.; others. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature Communications* **2020**, *11*, 1–15.
20. Stromatias, E.; Soto, M.; Serrano-Gotarredona, T.; others. An Event-Driven Classifier for Spiking Neural Networks Fed with Synthetic or Dynamic Vision Sensor Data. *Frontiers in Neuroscience* **2017**, *11*, 350. doi:10.3389/fnins.2017.00350.
21. Burrello, A.; Marchioni, A.; Brunelli, D.; others. Embedded Streaming Principal Components Analysis for Network Load Reduction in Structural Health Monitoring. *IEEE Internet of Things Journal* **2020**.
22. Ltd., R.P.T. Raspberry Pi Compute Module 3+ DATASHEET. https://www.raspberrypi.org/documentation/hardware/computemodule/datasheets/rpi_DATA_CM3plus_1p0.pdf, 2019.
23. HiveMQ. HiveMQ Documentation V4.5. <https://www.hivemq.com/docs/hivemq/4.5/user-guide/introduction.html>, 2019.
24. Tokognon, C.A.; Gao, B.; Tian, G.Y.; others. Structural health monitoring framework based on Internet of Things: A survey. *IEEE Internet of Things Journal* **2017**, *4*, 619–635.
25. Moradi, S.; Qiao, N.; Stefanini, F.; others. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). *IEEE transactions on biomedical circuits and systems* **2017**, *12*, 106–122.
26. Schemmel, J.; Fieres, J.; Meier, K. Wafer-scale integration of analog neural networks. 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence). Ieee, 2008, pp. 431–438.
27. Benjamin, B.V.; Gao, P.; McQuinn, E.; others. Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations. *Proceedings of the IEEE* **2014**, *102*, 699–716.

28. Davies, M.; Srinivasa, N.; Lin, T.H.; others. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* **2018**, *38*, 82–99.
29. DeBole, M.V.; Taba, B.; Amir, A.; others. TrueNorth: Accelerating From Zero to 64 Million Neurons in 10 Years. *Computer* **2019**, *52*, 20–29. doi:10.1109/mc.2019.2903009.
30. Kim, J.; Caire, G.; Molisch, A.F. Quality-aware streaming and scheduling for device-to-device video delivery. *IEEE/ACM Transactions on Networking* **2015**, *24*, 2319–2331.
31. Ponulak, F.; Kasinski, A. Introduction to spiking neural networks: Information processing, learning and applications. *Acta neurobiologiae experimentalis* **2011**, *71*, 409–433.
32. Barchi, F.; Zanatta, L.; Parisi, E.; others. An Automatic Battery Recharge and Condition Monitoring System for Autonomous Drones. 2021 IEEE International Workshop on Metrology for Industry 4.0 IoT, 2021.
33. Instruments, N. PXI Systems. <https://www.ni.com/it-it/shop/pxi.html>, 2021.