Neural Computing and Applications (2024) 36:18897–18917
https://doi.org/10.1007/s00521-024-10191-5

ORIGINAL ARTICLE

Energy efficient and low-latency spiking neural networks on embedded microcontrollers through spiking activity tuning

Francesco Barchi[1] · Emanuele Parisi[1] · Luca Zanatta[1] · Andrea Bartolini[1] · Andrea Acquaviva[1]

Received: 15 November 2023 / Accepted: 5 July 2024 / Published online: 1 August 2024

Abstract
In this work, we target the efficient implementation of spiking neural networks (SNNs) for low-power and low-latency applications. In particular, we propose a methodology for tuning SNN spiking activity with the objective of reducing computation cycles and energy consumption. We performed an analysis to devise key hyper-parameters, and then we show the results of tuning such parameters to obtain a low-latency and low-energy embedded LSNN (eLSNN) implementation. We demonstrate that it is possible to adapt the firing rate so that the samples belonging to the most frequent class are processed with less spikes. We implemented the eLSNN on a microcontroller-based sensor node and we evaluated its performance and energy consumption using a structural health monitoring application processing a stream of vibrations for damage detection (i.e. binary classification). We obtained a cycle count reduction of 25% and an energy reduction of 22% with respect to a baseline implementation. We also demonstrate that our methodology is applicable to a multi-class scenario, showing that we can reduce spiking activity between 68 and 85% at iso-accuracy.

Keywords Spiking NN · Low-power · Microcontroller · Edge computing · SHM · Energy efficiency · Cyber-physical systems

1 Introduction

Recently, the research community has shown increased interest in the third generation of artificial neural networks (ANNs), known as spiking neural networks (SNNs) [1]. This interest spans various embedded application domains, including anomaly detection and autonomous cyber-physical systems (CPS) [2, 3]. Their brain-inspired, event-based nature has the potential to significantly reduce energy requirements compared to traditional ANNs [4–10].

Also, the dynamic neuron structure of spiking neural networks makes them suitable for microcontroller-based applications that involve the processing of sensor data generated by sparse physical events occurring over time [11, 12] or produced by neuromorphic sensors such as the retina or cochlea: These sensors generate signals in the form of spikes in response to changes in physical quantities, instead of performing regular sampling [13].

However, SNNs are not used uniquely with event-based input. Indeed, recent research works have demonstrated that converting streaming input signals into spectral coefficients is also effective [14].

Additionally, also due to the lack of consolidated benchmarks with dynamic signals, SNNs are typically evaluated (and compared) with static input, such as persistent images, where pixel values are converted into spikes using either rate or temporal coding. In the first case, the values are proportional to the spike rate, and in the second time intervals between spikes. In this paper, we consider both dynamic and static input, where dynamic input comes

✉ Andrea Acquaviva
andrea.acquaviva@unibo.it

Francesco Barchi
francesco.barchi@unibo.it

Emanuele Parisi
emanuele.parisi@unibo.it

Luca Zanatta
luca.zanatta3@unibo.it

Andrea Bartolini
a.bartolini@unibo.it

[1] Department of Electrical, Electronic and Information Engineering - DEI, Università di Bologna, Viale del Risorgimento 2, 40050 Bologna, Italy

from a real application of anomaly detection. The static benchmark instead is a handwritten digits dataset used in many SNN related works [15–18].

As the target architecture for SNN execution, in this work, we focus on a low-cost commercial MCU equipped in an industrial IoT sensor node designed for anomaly detection tasks. However, the general approach we propose for spiking activity tuning is not limited to a specific implementation and can be applied to custom neuromorphic accelerators [19–23].

The primary distinction between SNNs and ANNs lies in their utilization of spikes for both encoding and information processing. As such, spiking activity is a pivotal parameter to be explored when designing an efficient network.

Spiking activity is represented by the number of spikes that have to be processed in the inner layers of the network for a given input sample. Spikes are generated when a neuron in the network "fires". Recurrent SNNs, also called LSNN (long short-term SNN), are typically composed by a single hidden neuron layer with feed-back connections, differently from feed-forward networks. This makes the prediction of "firings" non-trivial because it depends on time-dependent neuron internal states. On the other side, LSNNs are interesting because they allow efficient processing of temporal data streams from sensors (similarly to their artificial counterparts such as LSTMs). Moreover, with respect to feed-forward SNN, they are more powerful as they can exploit the full potential of SNN [1, 11, 24].

To enable the implementation of brain-inspired, edge processing on low-power, resource-constrained microcontrollers, the exploration of spiking activity within network models is a key aspect. For instance, SNNs have recently garnered attention in embedded applications such as object recognition and obstacle detection for camera-based or radar-based autonomous robot tasks [25, 26]. In these domains, the reduction of spiking activity and minimization of latency are relevant aspects.

The aim of this work is to leverage spiking activity to enhance the energy and computational efficiency of SNN implementations, as spike processing directly correlates with the number of cycles and, consequently, the energy consumption of the target device. Furthermore, the latency of SNN inference is correlated with spiking activity. In contrast to their artificial counterparts, SNNs, operating as dynamical systems, necessitate the processing of spikes over a predefined period (specified as a parameter) to reach an output class convergence.

We introduce a methodology designed for training LSNNs (while being applicable to non-recurrent SNNs) with the aim of fine-tuning spiking activity for specific classes. This approach results in the creation of "asymmetric" network models, where one class exhibits reduced spiking activity compared to others.

Furthermore, we illustrate that in the most extreme scenario, where a desired class exhibits zero spiking activity, it is feasible to implement a low-latency network with significantly diminished computational overhead. Our experiments reveal that in the context of the anomaly detection application for which we deploy our LSNN model, achieving this extreme scenario is attainable.

In this paper, we employ a combination of two strategies:

(i) Through an exploration of LSNN models, we identify critical network hyper-parameters that influence spiking activity. Then, we devise tuning policies to reduce this activity while maintaining acceptable classification accuracy.
(ii) We introduce class imbalance (i.e. a disparity in the number of training samples belonging to different classes) during the training process. This deliberate imbalance is leveraged to fine-tune hyper-parameters, resulting in the creation of models where certain classes exhibit minimal or, in some cases, zero neuron firing. This controlled approach allows us to dictate which class experiences a reduced firing rate, leading to an "asymmetric" network behaviour.

This approach proves effective in reducing energy consumption and latency for classes that are more frequently represented in the input data. For example, in the context of anomaly detection, input samples belonging to the "normal" class are more prevalent than those belonging to the "anomaly" class. By reducing network firing activity for the normal class, we achieve an overall reduction in energy consumption and latency compared to a network with a more uniform distribution of spiking activity.

Compared to the original models, the derived models exhibit decreased execution cycles and reduced energy consumption when subjected to samples associated with low-firing classes during the inference phase. It's important to note that while the proposed method can be employed in conjunction with SNN quantization techniques, an in-depth investigation of the quantization impact falls beyond the scope of the current study.

Leveraging the introduced tuning approach, we design an embedded LSNN implementation tailored for low-power microcontrollers, referred to as eLSNN. In case the tuning process successfully identifies an asymmetric model that achieves the extreme scenario of zero spiking activity for a particular class, we used it to develop an optimized variant, denoted as reduced eLSNN. This optimized version further reduces both the number of cycles and energy

consumption by eliminating certain algorithmic operations compared to a complete eLSNN implementation. We argue that if such a condition is attainable for a specific task, it can be harnessed through the proposed spiking activity tuning strategy.

To demonstrate the effectiveness of the proposed methodology, we applied the tuning procedure to both binary and multi-class classification scenarios. In the case of binary classification, we present performance and energy results obtained from an actual hardware sensor node used for structural health monitoring (SHM) applications. The dataset utilized in this evaluation comprises accelerometer data collected from a highway viaduct under both healthy and damaged conditions. This dataset was used in other state-of-the-art studies on SHM data analysis [3, 27]. In particular, in [3] SNNs were successfully applied to damage recognition. Another application of SNNs to SHM can be reported in [28], where a feed-forward SNN emulated on a neuromorphic accelerator is proposed.

For the multi-class scenario, we employed the widely recognized MNIST handwritten digit database [29], widely adopted for SNN training. Our selection of this dataset aims to demonstrate the feasibility of our proposed method in a multi-class context, rather than evaluating the performance of LSNNs on this benchmark, which is the focus of other papers in the literature.

Compared to existing state-of-the-art approaches that primarily focus on reducing latency and energy consumption in SNNs, the contributions of this paper are:

- We introduce a novel methodology for leveraging network and dataset hyper-parameters to fine-tune LSNN spiking activity directly within the trained LSNN models.
- We demonstrate a practical approach to control the firing activity of specific classes through a training methodology that results in the creation of asymmetric network models.
- We present an optimized computational model, referred to as "reduced", which minimizes latency and computational overhead when zero firing activity for a specific class is attainable.
- We provide an optimized implementation on a low-power microcontroller-based sensor node, enabling quantitative performance and energy evaluations.
- Our work demonstrates how the tuning strategy effectively reduces spiking activity during inference for both binary and multi-class classification scenarios.
- We report energy and latency measurements of the "reduced" model implementation on the sensor node performing an anomaly detection task.

Experimental results illustrate the effects of the spiking activity tuning for both multi-class and binary scenarios. In the multi-class scenario, we observed a reduction in spiking activity ranging from 68 to 85% for specific classes in comparison to the average activity levels.

In the context of the SHM application, we conducted cycle and power measurements using a hardware implementation. This setup involved a Hardware-in-the-Loop approach, allowing us to inject real accelerometer data into the sensor node.

The results obtained on the reduced version show a 25% reduction in cycle count and a 22% decrease in energy consumption when compared to the complete eLSNN version.

The rest of this paper is organized as follows. Section 2 presents some background on LSNN and the adopted neuron model. Section 3 explains the efficient LSNN implementation for microcontroller targets. Section 3.2 explains the eLSNN tuning strategies. Section 5 reports the experimental test bed and results, and Sect. 6 concludes the work.

2 Background and related works

2.1 Neuron model

ANNs are computing systems inspired by the biological neural networks that constitute animal brains. The base unit of the ANNs is the neuron that has a differentiable activation function. This feature allows training, with gradient-based optimisation methods and backpropagation. The SNN model is a brain-inspired (so-called third generation) type of neural network. They have a greater computational capacity as the single neuron is modelled with a much more complex dynamic than the neurons present in traditional artificial neural networks (ANNs). This means that SNNs can solve the same tasks as ANNs with fewer neurons [1]. Moreover, their hardware implementation on neuromorphic architectures and accelerators can lead to greater energy efficiency in data management and computation [19–23]. In this work, we do not focus on neuromorphic implementation, but rather on commercially available MCUs widely used for embedded and cyber-physical applications. However, the proposed LSNN model exploration and tuning methodology is applicable also to dedicated hardware architectures, provided a suitable training algorithm is available.

SNNs differ from the ANNs in that the neuron is a dynamic system composed of differential equations that describe physical metrics like membrane potential and synapse injecting currents. A spiking neuron is characterized by a dynamic behaviour described by the evolution of

an internal state $\boldsymbol{h}_j^t$ eventually triggering output spikes (binary variables), representing the observable state z_j^t [14].

Many neuron models can be found in literature: Hodgkin–Huxley (HH) [30], Izhikevich (IZK) [31], Integrate and Fire (LIF) [32]. Each of them implements a different dynamic behaviour, leading to a different trade-off between biological plausibility and computational cost.

Within an SNN, the connections between neurons are mediated by synapses. Each synapse transports information from a pre-synaptic neuron i to a post-synaptic neuron j. The various neuron models describe, with different degrees of approximation, the evolution of the membrane potential stimulated by the currents injected by synapses. When the membrane potential reaches a threshold value, the neuron emits an impulse called a "spike". The biological plausibility of a spiking neuron lies, among other things, in its ability to model different operating states:

- *Rest period* The neuron is waiting to receive a stimulus strong enough to emit a spike (i.e. it fires);
- *Absolute refractory period* In this state, the neuron cannot fire independently on the input activity;
- *Relative refractory period* The neuron can fire, but it has to receive a stimulus stronger than the one that needs to fire in the rest period.

In this manuscript, we focus on LSNN [14]. A LSNN network is composed of an input layer composed of I sources, which is in turn connected in an all-to-all fashion to a recurrent layer composed of N adaptive-integrate and fire neurons (ALIF) through a $W^I \in \mathbb{R}^{N,I}$ matrix. The hidden layer is recursively connected to itself with an all-to-all connection matrix $W^H \in \mathbb{R}^{N,N}$. Lastly, the hidden layer will be all-to-all connected to the output layer composed of O neurons. This connection creates one more connection matrix $W^O \in \mathbb{R}^{O,M}$.

In the ALIF neuron model, the hidden state h_j is represented by the membrane potential v_j and by a variable a_j, which controls the dynamic behaviour of the threshold voltage $v_t h$.

In our network model, the output is represented by the membrane potential of the neurons in the output layer, which is described by the following equation:

$$y_k^t = \overbrace{e^{-\frac{\delta t}{\tau_o}}}^{\beta} y_k^{t-1} + \overbrace{\sum_j W_{kj}^O z_j^t}^{\text{ALIF}\rightarrow\text{Out}} + \overbrace{b_k}^{\text{Bias}} \tag{1}$$

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

The training of the network parameters for a given learning task requires finding the optimal parameters of the LSNN networks which are t_{inf} that is the number of integration ticks (δt) that the network uses to process the network, I, N, τ_m, τ_o, v_{thc} that is used to compute v_{th} ($v_{th} = v_{thc}/(1 - e^{-\delta t/\tau_m})$) and τ_{ac} that is used to compute τ_a ($\tau_a = \tau_{ac} t_{inf}$). Given that a spike neuron model is considerably more complex than an artificial neuron model (accumulation and threshold), its training requires higher computational effort than its simplified version. To train the LSNN model, we applied the backpropagation through time (BPTT) algorithm [14, 33].

2.2 Structural health monitoring scenario

The dataset has been collected from a highway viaduct, representative of many SHM monitoring systems mounted on other viaducts, such as the one presented in [27]. The viaduct is built with eighteen sections, each supported by two pairs of concrete pillars at their ends. The dataset refers to a single section instrumented for data analysis before a scheduled maintenance intervention in this work. The maintenance was necessary for the strengthening of the viaduct structure. The dataset was made of accelerations, which were segmented to isolate the crossing vehicles that will form the dataset. Each vehicle was labelled with the label belonging to the class "damaged bridge" or "healthy bridge", depending on whether the vehicle had crossed the bridge. This procedure resulted in a dataset of 6394 training samples, 546 validation samples and 994 test samples. The dataset is balanced. The fast Fourier transform (FFT) of each sample was computed. The FFT will be the input of the network and will be kept constant for the inference time. This approach has been successfully applied to tasks dealing with similar input and network models like in [14] for phoneme classification and in [25] for image classification.

2.3 Related works

Very recently, given the rising interest in SNN, researchers focused on the latency problem, which is correlated to spiking activity, including hardware accelerators to enable real-time applications [16]. Indeed, a network trained for low latency also shows reduced spiking activity. A summary of basic approaches is present in [34]. The majority of literature work addressing low latency focuses on training ANN networks and then converting to SNN while optimizing latency. This typically limits the SNN network structure to feed-forward networks with simple neuron models (i.e. LIF). A recent approach aims at capturing pre-activation values distribution trying to minimize the error between ANN and converted SNN, resulting in SNN feature ultralow-latency and activation sparsity [35].

Compared to these approaches, in this work, we focus on directly trained SNN. The proposed method is designed to directly train SNN, as it exploits a specific loss term in conjunction with dataset unbalancing. In general, it is recognized that direct training leads to a lower firing activity with respect to covered SNNs, thus leading to higher energy efficiency.

More recently, in [36], a training framework for one-time-step SNNs that uses a novel regularization approach is presented, showing a reduction of spiking activity. Authors use pre-trained dynamic neural network (DNN) models, and they do not focus on firing control for a given class. In [37], the authors implement a decision-making agent in order to minimize the timesteps in an interactive manner. In [38], a method to generate high-performance, low-latency (up to 32-time steps) SNNs converted from ANN is proposed, exploiting an optimized initialization of membrane potential. In [39], a shifting of the initial membrane potential is adopted to calibrate the output spikes and obtain high accuracy while reducing latency for converted ANN-SNN models. Similarly to the previous methods, in [18], latency is reduced using a pre-charged membrane potential (PCMP). An Iterative Initialization and Retraining method for SNNs (IIR-SNN) is proposed in [34] to perform single-shot inference. The authors focus again on converted ANN-SNN and feed-forward networks.

An alternative method based on information representation is presented in [40], where the differentiation of spike representation (DSR) is proposed to achieve low latency and high performance. This method is demonstrated to achieve comparable or higher accuracy than ANN even with low-latency networks (with a few tens of time steps). In [41], a hybrid training method is presented based first on ANN-to-SNN conversion to select initial thresholds in each network layer. Successively, a gradient descent-based training method is used that learns the correct membrane leak and firing threshold for each layer of a deep spiking network via error backpropagation. The tailored membrane leak and threshold for each layer leads to large improvements in activation sparsity and energy efficiency. All these methods are applied to ANN-to-SNN converted networks, while in our paper, we perform direct training using BPTT. Also, our method focuses on spiking activity tuning rather than timestep reduction.

Compared to these approaches, the method proposed in this work is original in that it is the first attempt to exploit asymmetry in the dataset coupled with a loss term to adapt the firing rate. Overall, the advantage is that, in a classification task, it is possible to decide which class is associated with a lower firing rate. In case a class is less represented during inference, this actually leads to an overall lower firing rate, thus reducing energy consumption. Note that reducing the timesteps is not the same as reducing the spiking activity. Indeed, in a single timestep, the network can integrate many spikes. From this perspective, our approach is orthogonal to training methods focused on timesteps reduction. From the other side, our method can be integrated to these ones to further reduce the overall spiking activity.

Concerning fully directly trained SNN methods, Li et al. [42] introduces SEENN-I and SEENN-II. The former utilizes confidence score thresholding to determine the optimal number of timesteps, whereas the latter employs a policy network to predict the number of timesteps. In [43], the authors propose a Shrinking SNN (SSNN) in which, during the training, they reduce the number of timesteps over multiple training steps. In [17], the authors introduce a novel temporal batch normalization through time (BNTT) to solve the issue of unstable training. This work also shows that trained networks are also characterized by lower inference latency with respect to original training. All these methods reduce time steps as a side effect, while our method focuses on a direct tuning of spiking activity with control of spiking for a given class. A novel approach based on temporal pruning done on training iterations with a reduction of timesteps at each iteration is proposed in [44]. This approach is focused on timestep reduction and works on recurrent SNN. Compared to the method proposed in this work, it focuses directly on timestep reduction, while we selectively reduce spiking activity on specific classes and we obtain latency reduction as a side effect. Indeed, their try to overcome the problem of too poor spiking activity playing on the neuron thresholds. In our work, the training approach used in our work (which leverages the one presented in [14]) allows us to exploit the f^* parameter to directly control spiking activity.

The advantage of being able to tune spiking activity materializes in tasks where there is an asymmetry in the class representation during inference, which is samples belonging to one class are more represented than others. One such example is the case of anomaly detection task, where the "normal" class is typically more represented than the "anomalous" class. In these cases, our method outperforms other approaches in terms of energy consumption. Also, as mentioned, our approach can be coupled with any other training method, including the ones aimed at reducing the timesteps. In other cases, where this asymmetry is not present, the proposed approach does not introduce any predictable advantage. Another consideration that we added in this discussion is related to the proposed approach's impact on latency, as other state-of-art papers deal with latency, presenting training methods to reduce the number of timesteps. First of all, our approach is not an alternative, as it can be applied orthogonally to introduce asymmetry in the spiking activity related to the most represented class. From the other side, our method

can further reduce latency, in the extreme case where the spiking activity is reduced to zero for a specific class. From this perspective, the proposed approach outperforms other methods as no one leads to a minimum of zero timesteps. Clearly, this is an extreme case and relates only to one class. As said, this is convenient only in case there is an asymmetry in the class representation during inference: for those samples belonging to the most represented class, we can reach very low or even null spiking activity for the large majority of the samples. As discussed in the experiments, the number of samples with reduced spiking activity can be regulated by an asymmetry threshold as a training hyper-parameter.

3 Efficient embedded LSNN model exploration methodology

In this section, we introduce the key hyper-parameters that we leverage to tune the spiking activity of the neurons and we describe the tuning methodology we used on the training dataset for reducing spiking activity of specific classes. The key idea is to control which class will have low-firing activity in the output layer to reduce both energy and latency. Indeed, if the more frequent class during inference causes low-firing activity, we obtain a larger impact on energy reduction. This is typically the case of anomaly detection applications, where the large majority of input samples represent "normal" conditions, while "anomaly" conditions are less frequent. If we force the normal condition to cause less firing activity in the network, then we will save more energy (considering the application lifetime) with respect to a network that has a more uniform distribution of firing activity across the classes.

3.1 Loss function impact on spiking activity

In this manuscript, we propose to leverage a training hyper-parameter to obtain a low activity. Bellec et al. [14] introduce two regularization parameters in computing the loss of the network: (i) the classification loss (L_p) computed with the cross-entropy and (ii) the firing rate loss (L_f). The L_f is computed as the difference between the firing rate activity of the network and the value f^* that is the desired firing rate [14].

$$L_f = \frac{1}{2}\sum_j \left(\frac{1}{nT}\sum_{t=1}^{nT} z_j^t - f^*\right)^2 \tag{2}$$

The overall loss is the weighted sum of the classification loss and the firing rate loss.

$$L = \kappa_p L_p + \kappa_f L_f \tag{3}$$

The coefficients κ_p and κ_f amplify or dampen the contribution of loss values. We explore these parameters in the grid search performed in the multi-class scenario in the experimental section.

We also explore the class unbalancing ratio in the training dataset ($u_{\%}$) to amplify the f^* parameter effect in the loss function and tune the firing activity of a specific class in the binary classification scenario of anomaly detection.

3.2 Training for reducing spiking activity

In this section, we present a novel methodology to obtain LSNN models which can run efficiently on resource-constrained embedded systems. We call these models eLSNN. As early introduced in the previous section we can identify three sets of parameters which impact to the cost of the proposed eLSNN model inference: i) The number of neurons in the input and hidden layer (I, N); ii) The inference time (t_{inf}) and iii) The firing activity (z). The first two are properties of the LSNN model and do not depend on the input sample for which the LSNN is computing the inference. The third instead depends on the latent representation associated with the given input sample. As described in Sect. 3.1 a given LSNN model can be tuned to reduce the overall firing rate (though the f^* training hyper-parameter), and to zero the hidden layer firing activity for input samples belonging to a specific class. In this manuscript, we propose to control the specific class which can be learnt with zero firing rate by controlling the class unbalance in the training dataset. In anomaly detection applications, where the anomalous class is rare, having a LSNN model which represents the normal class with zero firing activity can further reduce the inference cost.

Figure 1 shows the proposed eLSNN tuning methodology. It consists of the following main steps:

1. The training set is processed to obtain a new training set with an artificial class unbalance which induces asymmetric firing behaviour for the input samples associated with the most represented class. For a binary classification problem, we define the dataset unbalance ($u_{\%}$) as the percentage of a given class in the new training set. We refer to $u^0_{\%}$, for instance, in case the wanted class is class "0". In the experiment section, we will show the impact of this parameter. Also, when imposing an overall spiking activity reduction (see Step 2), by increasing the fraction of samples in the training set for a given class, that class will reduce its spiking activity. The spiking activity is thus inversely

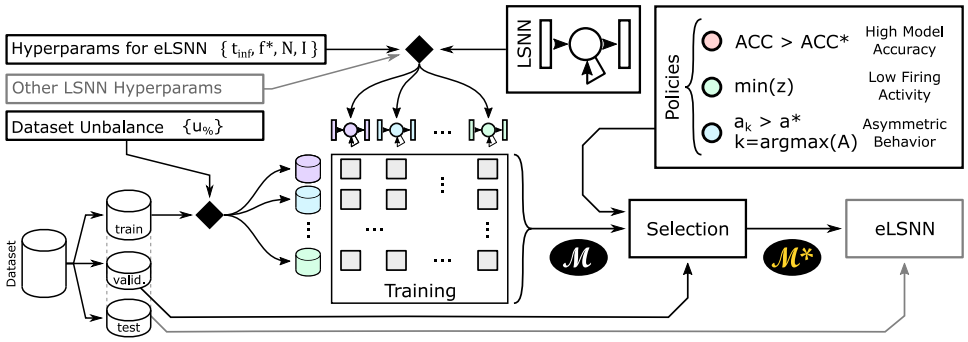

Fig. 1 The eLSNNs tuning methodology. It consists of four steps: (i) Splitting the dataset, (ii) Applying the dataset unbalance to obtain multiple training set instances, (iii) Training models in the grid search using the eLSNN hyper-parameters, (iv) Using a set of policies, identifying the best configuration for the network implementation on the microcontroller

proportional to the fraction of samples in the training set.

2. A population of trained LSNN models ($\{\mathcal{M}\}$) is generated with a grid search on the eLSNN hyper-parameters. The eLSNNs hyper-parameters we focus on in this paper are I, N, t_{inf}, and f^*. Other hyper-parameters do not directly influence the computational complexity of the LSNN model but are included in the grid search to achieve higher model accuracy.
3. From the population of trained models ($\mathcal{M}$) we select a set of optimal models ($\mathcal{M}^*$), using the following criteria (in logical AND) on the validation set:
   (a) Accuracy above a specified threshold;
   (b) Low neuronal activity;
   (c) Asymmetric behaviour.

The asymmetric ratio for a class (a_i) we refer to in Step 2 is defined as the fraction of input samples classified by the model in a given class with a fire rate equal to zero among all the sample classified by the model in the class:

$$a_i = |\{s : M(s) = i, Z(s, M) = 0, s \in D\}| / |\{s : M(s) = i, s \in D\}| \tag{4}$$

In Eq. 4 i is the class, s is the sample and D is the dataset. We employ the optimized asymmetric eLSNN model inference only if the highest asymmetric ratio (a_k) for the class k is above a user-defined threshold (a^*). We will explore this trade-off in the experimental result section. Formally $A : \{a_1, a_2, ...a_O\}$ (used also in Fig. 1) is the vector of asymmetric ratios a_i computed for each class.

The class with the maximum asymmetric ratio is the class $k = argmax(A)$ and if $a_k > a^*$ we can consider the model as an asymmetric model. From optimal models, we select to be deployed on the resource-constrained micro-controller. The optimized implementation is described in Sect. 4, while in Sect. 5, we report details about accuracy, energy and latency on the test set where we will discuss the impact of the application of the proposed methodology to binary and multi-class classification scenarios.

4 eLSNN implementation

In this section, we first describe a baseline embedded LSNN model which targets resource-constrained micro-controllers and implements the ALIF neuron model. Then, we describe an optimized LSNN implementation which exploits the asymmetry in the latent representation in order to skip processing cycles during inference. Figure 2 depicts the LSNN computation.

The network update consists of four main steps: (i) Membrane voltage update ($\boldsymbol{v}$), (ii) Membrane voltage threshold update ($\boldsymbol{a}$), (iii) Evaluation of the spike emission ($\boldsymbol{z}$) (iv) Evaluation of the output ($\boldsymbol{y}$). The diagram shows $\boldsymbol{x}$ data dependencies in green, $\boldsymbol{v}$ data dependencies in blue, $\boldsymbol{a}$ data dependencies in red, $\boldsymbol{z}$ data dependencies in purple, and $\boldsymbol{y}$ data dependencies in magenta.

4.1 Neuron model

The evolution of the membrane potential v_j of neuron j over time is given by the following equation:

$$v_j^t = \overbrace{e^{-\frac{\delta t}{\tau_m}}}^{\alpha} v_j^{t-1} + \overbrace{\sum_{i \neq j} W_{ji}^H z_i^{t-1}}^{\text{ALIF}\rightarrow\text{ALIF}} + \overbrace{\sum_n W_{jn}^I x_i^t}^{\text{In}\rightarrow\text{ALIF}} - \overbrace{v_{th} z_j^{t-1}}^{\text{Reset}} \tag{5}$$

In 5, we considered the hidden state as $\boldsymbol{h_j^t} := [v_j^t, a_j^t]$ and the observable state as $z_j^t \in \{0, 1\}$ [14]. v is called membrane potential; it increases when the neuron receives a stimulus (spike or current). According to [14], in this model, describing the behaviour of internal neurons, biases are

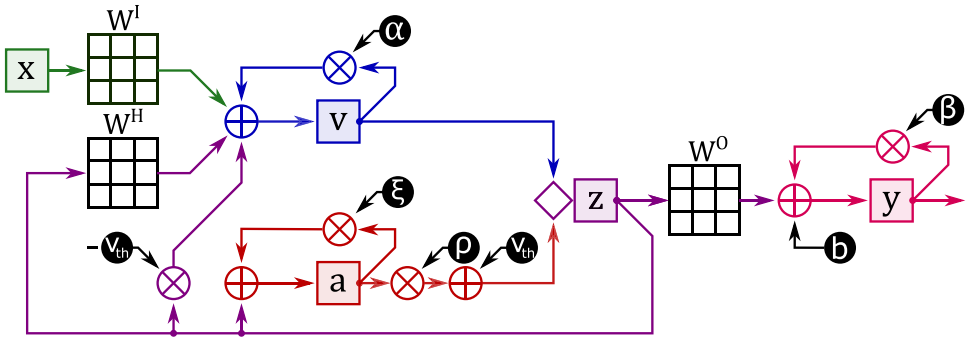

Fig. 2 LSNN update cycle flowchart. Network status variables are represented with rectangles. Operations are represented with circles. Matrix multiplication operations with synaptic weights are represented by grids. Constants are represented with black circles. Data movements are represented by arrows between operators and operands

absent. We instead introduce biases in the output neurons model as shown in Eq. 1.

When v reaches a threshold called v_{th}, it emits a spike z. In the case of ALIF neurons, the variable v_{th} is dynamic: It increases when a spike is emitted and decreases in the absence of spikes.

This behaviour is described by the variable a_j for the $j-th$ neuron as follows:

$$a_j^t = \overbrace{e^{-\frac{\delta t}{\tau_a}}}^{\xi} a_j^{t-1} + z_j^{t-1} \tag{6}$$

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

$$z_j^t = \begin{cases} 1 & \text{if } v_j^t \geq A_j^t \text{ and } r_j^t \neq 1 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

In Eq. 5, the first term accounts for the decay of the membrane potential (α), which depends on the membrane time constant τ_m and the tick (δt) of the network.

The second and the third terms describe the contribution of the recurrent part and the input layer, respectively. x represents the input signal and can be either spikes or currents. The final term in Eq. 5 accounts for the reset of the membrane potential upon crossing v_{th}. Due to this term, v is a non-differentiable function due to the spike emission. In ALIF neurons, additional equations are used to describe the dynamics of the threshold voltage update. More specifically, in Eq. 6 ξ is the decay of the adaptive threshold, and it depends on δt and τ_a called decay time constant. a is weighted by a factor ρ before being added to v_{th}. Equation 7 describes the firing behaviour, similar to LIF neurons. The neuron can spike (fire) only if it reaches a certain value (A) and it is out from the refractory period ($\boldsymbol{r}^t \neq 1$). The refractory period is a time interval in which the neuron is insensitive and happens right after the spike emission.

To train the LSNN model, we applied the backpropagation through time (BPTT) algorithm [14]. This algorithm is a gradient-based training model that is normally applied to the recurrent network whose goal is to minimize the difference between the network output compared to a target (ground truth) by updating the weights of the network. To achieve this task, the networks have to be unrolled. Since Eq. 5 is a discontinuous function, for applying the BPTT, we can use a pseudo-derivative that allows bypassing this problem [14]:

$$\psi_j^t = \frac{\partial z_j^t}{\partial v_j^t} = \begin{cases} \frac{\gamma_{pd}}{v_{th}} \max\left(0, 1 - \left|\frac{v_j^t - A_j^t}{v_{th}}\right|\right) & \text{if } r_j^t \neq 1 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

The above equation represents the distance between the spike threshold and the v level. The default value of γ is 0.3.

4.2 Membrane voltage update

The computation involved in the membrane voltage update is done through the following steps:

1. $\boldsymbol{v}^{(\alpha)} := \alpha \boldsymbol{v}$—Membrane voltage decay
2. $\boldsymbol{v}^{(I)} := W^I \boldsymbol{x}$ —Contribution of inputs on membrane voltage
3. $\boldsymbol{v}^{(H)} := W^H \boldsymbol{z}$ —Contribution of internal neuronal activity on membrane voltage
4. $\boldsymbol{v}^{(th)} := -v_{th} \boldsymbol{z}$ —Reset of membrane voltage in case of previous spike emission
5. $\boldsymbol{v} \leftarrow \boldsymbol{v} + \boldsymbol{v}^{(\alpha)} + \boldsymbol{v}^{(I)} + \boldsymbol{v}^{(H)} + \boldsymbol{v}^{(th)}$ —Update of membrane voltage

In the membrane voltage update step, for the internal neuron's contribution the elements of $\boldsymbol{z}$ can only assume binary values ($z_i \in \{0, 1\}$), the $W^H \boldsymbol{z}$ operation can be implemented using only additions. As shown in Fig. 1 on the left, for a naive implementation, the number of operations to be performed is $(I + N + 2)Nt_{inf}$ additions and $(I + N + 2)Nt_{inf}$ multiplications. The required operations

will depend on the number of nonzero elements in the vector $\boldsymbol{z}$. The number of nonzero elements in the vector $\boldsymbol{z}$ will change with each inference tick. We then identify using the symbol $\tilde{z}$ the average number of nonzero elements of $\boldsymbol{z}$ for each inference tick. The value of $\tilde{z}$ depends on the behaviour of the network. When training the network, it is possible to minimize $\tilde{z}$ by introducing its value into the loss calculation. In Table 1 on the right, we report the operations (additions and multiplications) of the optimized implementation for each component of $\boldsymbol{v}$. In the optimized implementation, the computation of $\boldsymbol{v}^{(H)}$ requires $\tilde{z}$-1 additions.

The computation steps of our LSNN implementation are summarized in Algorithm 1.

Algorithm 1 Pseudo-code of the eLSNN timestep computation.

$N_t :=$ Number of timesteps
$N_n :=$ Number of neurons
$\alpha :=$ Membrane voltage decay factor
$\xi :=$ Adaptive threshold decay factor
$\rho :=$ Adaptive threshold correction factor
$\beta :=$ Output membrane voltage decay factor
for $t \leq N_t$ **do**
  $\boldsymbol{v}^t \leftarrow \alpha \boldsymbol{v}^{t-1} + \boldsymbol{i}^t \cdot \boldsymbol{W}_I + \boldsymbol{z}^{t-1} \cdot \boldsymbol{W}_H - v_{th} \boldsymbol{z}^{t-1}$ ▷ Membrane voltage update
  $\boldsymbol{a}^t \leftarrow \xi \boldsymbol{a}^{t-1} + \boldsymbol{z}^{t-1}$ ▷ Adaptive threshold update
  $\boldsymbol{A}^t \leftarrow v_{th} + \rho \boldsymbol{a}^t$
  $\boldsymbol{y}^t \leftarrow \beta \boldsymbol{y}^{t-1} + \boldsymbol{z}^t \cdot \boldsymbol{W}_O + b$ ▷ Output membrane voltage update
end for

In the case of current-based input, where the input vector $\boldsymbol{x}$ contains real values, the reduction of multiplications to additions concerns only the internal layers.

For instance, in [14], current input is used in phonemes recognition tasks performed over time windows on a speech waveform. Inspired by this work, we used the current input for the anomaly detection scenario, where a binary classification was applied to time windows on a waveform of vibrations. Following the approach in [14], the samples in a time window are pre-processed to extract spectral coefficients. This input is kept constant to the network for a period called *Exposure Time* t_{exp}. In what follows, we will refer to this eLSNN version as *current-driven* eLSNN, since it uses a continuous and constant input over the exposure time (that can be modelled as a current).

Within the network exposure time, the values of $\boldsymbol{x}$ are persistently presented to the network.

Table 1 Operations analysis for a single neuron

Item	Naive		Optimized	
	Sum	Mul	Sum	Mul
$\boldsymbol{v}^{(\alpha)}$	–	1	–	1
$\boldsymbol{v}^{(I)}$	I−1	I	$(I-1)/t_{exp}+1$	I/t_{exp}
$\boldsymbol{v}^{(H)}$	N−1	N	$\tilde{z}-1$	–
$\boldsymbol{v}^{(th)}$	–	1	1	–
$\boldsymbol{v}$	4	–	4	–
Total	I+N+2	I+N+2	$(I-1)/t_{exp}+\tilde{z}+5$	$I/t_{exp}+1$

These operations must be executed for each neuron (N) and for each inference tick (t_{inf})

4.2.1 Membrane voltage update: internal neurons contribution

The computation of $v^{(H)}$ requires a row-column multiplication between the matrix W^H (with $N \times N$ elements) and the column vector $\boldsymbol{z}$ (with $N \times 1$ elements). The result will be added to the vector $\boldsymbol{v}$. As shown in Fig. 1 on the left, for a naive implementation, the number of operations to be performed is $(I+N+2)Nt_{inf}$ additions and $(I+N+2)Nt_{inf}$ multiplications.

When the elements of $\boldsymbol{z}$ can only assume binary values ($z_i \in \{0,1\}$) the $W^H\boldsymbol{z}$ operation can be implemented using only additions). Let $\breve{z}$ be a list containing only the index of the nonzero elements of $\boldsymbol{z}$. By iterating over the elements of $\breve{z}$, we select the columns of W^H to be considered in the inner cycle (cycle of additions). The sum cycle iterates over the rows of the selected column and, for each element, it accumulates its content in the result vector. Therefore, the required operations will depend on the number of nonzero elements in the vector $\boldsymbol{z}$. The number of nonzero elements in the vector $\boldsymbol{z}$ will change with each inference tick. We then identify using the symbol $\tilde{z}$ the average number of nonzero elements of $\boldsymbol{z}$ for each inference tick. The value of $\tilde{z}$ depends on the behaviour of the network.

When training the network, it is possible to minimize $\tilde{z}$ by introducing its value into the loss calculation.

In Table 1 on the right, we report the operations (additions and multiplications) of the optimized implementation for each component of $\boldsymbol{v}$. In the optimized implementation the computation of $\boldsymbol{v}^{(H)}$ requires $\tilde{z}$-1 additions.

4.2.2 Membrane voltage update: input neurons contribution

In the case of current-based input, where the input vector $\boldsymbol{x}$ contains real values, the reduction of multiplications to additions concerns only the internal layers.

For instance, in [14], current input is used in phonemes recognition tasks performed over time windows on a speech waveform. Inspired by this work, we used the current input for the anomaly detection scenario, where a binary classification was applied to time windows on a waveform of vibrations. Following the approach in [14], the samples in a time window are pre-processed to extract spectral coefficients. This input is kept constant to the network for a period called *Exposure Time* t_{exp}. In what follows, we will refer to this eLSNN version as *current-driven* eLSNN, since it uses a continuous and constant input over the exposure time (that can be modelled as a current).

Within the network exposure time, the values of $\boldsymbol{x}$ are persistently presented to the network. This leads to the existence of three-time domains, represented in Fig. 3:

- Input time domain, accounting for input;
- Network time domain, accounting for the t_{exp} iterations performed by the network, possibly observing the same value of $\boldsymbol{x}$;
- Output time domain, accounting for output updates.

Figure 3 shows that vector $\boldsymbol{x}$ updates in sync with the input (input domain), while vector $\boldsymbol{v^{in}}$ is used to update $\boldsymbol{v}^{(I)}$ according to the network time steps. The number of operations to compute $\boldsymbol{v}^{(I)}$ then becomes $((I-1)/t_{exp}+1)Nt_{inf}$ additions and $(I/t_{exp})Nt_{inf}$ multiplications.

The second vector $\hat{\boldsymbol{z}}$ decouples the network time domain from the output. At each tick, the network will accumulate the emitted spikes inside the vector $\hat{\boldsymbol{z}}$.

In the output time domain, the content of the vector $\hat{\boldsymbol{z}}$ will be averaged over the exposure time, obtaining $\bar{\boldsymbol{z}}$, that will be used for the calculation of $\boldsymbol{y}^{(H)} := W^O\bar{\boldsymbol{z}}$.

The number of operations for computing $\boldsymbol{y}^{(H)}$ then becomes $(1/t_{exp}(N-1)O+\tilde{z})t_{inf}$ additions and $t_{inf}/t_{exp}NO$ multiplications.

Overall, in our implementation, as reported in Table 1, the number of additions required for the membrane voltage update is $((I-1)/t_{exp}+\tilde{z}+5)Nt_{inf}$ and the number of multiplications is $(I/t_{exp}+1)Nt_{inf}$.

4.2.3 Membrane voltage threshold update

This is a part of the computation characterizing the ALIF neuron model. Updating the maximum threshold for the membrane voltage is broken down into the following operations:

1. $\boldsymbol{a}^{(\xi)} := \xi\boldsymbol{a}$ - Evaluation of threshold decay
2. $\boldsymbol{a}^{(z)} := z$ - Threshold adjustment in case of previous spike
3. $\boldsymbol{a} \leftarrow \boldsymbol{a} + a^{(\xi)} + a^{(z)}$ - Threshold update
4. $a^{(\rho)} := \rho a$ - Weighted threshold computation
5. $v_{th}^{(a)} := v_{th} + a^{(\rho)}$ - Reference threshold augmentation

In total, $3Nt_{inf}$ additions and $2Nt_{inf}$ multiplications are required to implement the adaptive threshold functionality.

4.3 Evaluation of the spike emission

At each inference tick, the spike firing condition must be checked. For each neuron, the membrane voltage is compared with the spike threshold. In the ALIF neuron model, the threshold voltage is adapted to the activity of the neuron and increases as its activity increases. A spiking neuron is also inhibited for a period t_{ref} called refractory time. During this period, even if the neuron has a membrane voltage above the threshold, it will not fire. To check this condition, each neuron stores the information about the tick of the last spike $t_i^{(z)}$.

When checking the emission of a spike, two conditions must therefore be checked for each neuron:

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

If the above conditions are satisfied, the vector $\boldsymbol{z}$ will take the value "1" at position i, otherwise the value will be "0".

In our implementation, instead of directly handling the vector $\boldsymbol{z}$, we use the list of events $\breve{\boldsymbol{z}}$ to replace matrix multiplications $W^H\boldsymbol{z}$ and $W^O\boldsymbol{z}$ with additions. At the beginning of the spike emission check phase, the list $\breve{\boldsymbol{z}}$ of previous events is cleared. In the presence of a spike emission, the identifier of the spiking neuron will be added to the $\breve{\boldsymbol{z}}$ list.

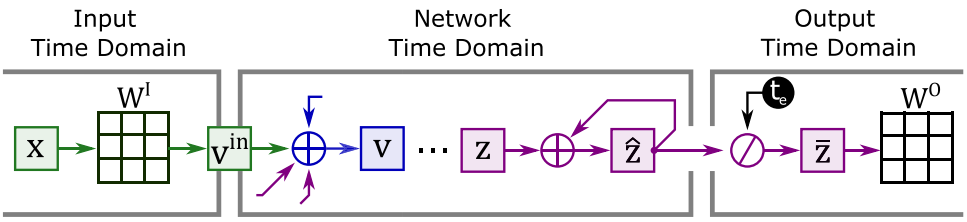

Fig. 3 Flowchart modified to handle current-driven input. In this application scenario, the input and output have two different time domains from the network. $\boldsymbol{v}^{in}$ and $\hat{z}$ vectors are used to pass information from the input to the network and from the network to the output

4.4 Evaluation of the output

Spikes generated by network neurons in the recurrent layer are received by output neurons at the same time as they are emitted. The output neurons have a similar update cycle to the recurrent neurons:

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

Again, the computation of $\boldsymbol{y}^{(\boldsymbol{H})} = W^O \boldsymbol{z}$ can be done using the list $\breve{\boldsymbol{z}}$ to perform $\tilde{z} Ot_{inf}$ additions instead of $(N-1)Ot_{inf}$ additions and NOt_{inf} multiplications of the naive implementation.

According to Eq. (5), in the presence of a constant input current each neuron in the internal layer receives a constant stimulus equal to $k_i = \sum_j W^{in}_{ji} x^t_j$. Considering initially null neuronal activity ($z^{t=0}_i = 0 \forall i$) it is possible to express the evolution of the membrane voltage by the following equation:

$$\begin{aligned} v^t_i &= \alpha^t v_0 + k_i \sum_{\tau=0}^{t-1} \alpha^\tau \\ &= \alpha^t v_0 + k_i \frac{1-\alpha^t}{1-\alpha} \end{aligned} \tag{9}$$

If the inference time of an eLSNN is known and is equal to T it is possible to identify the range of values k_i that given as input to a neuron do not cause the emission of a spike:

$$\begin{aligned} v^T_i &< v_{th} \\ \alpha^T v_0 + k_i \frac{1-\alpha^T}{1-\alpha} &< v_{th} \\ k_i &< \underbrace{v_{th} \frac{1-\alpha}{1-\alpha^T}}_{k^*} - \underbrace{v_0 \alpha^T \frac{1-\alpha}{1-\alpha^T}}_{k^{(*,0)}} \end{aligned} \tag{10}$$

In the case of asymmetric networks, in case we are able to train an asymmetric class to be recognized with no spikes (for instance the most represented one in input), this class, identifiable by the absence of synaptic activity, can be detected without performing the network calculation, without waiting for $t_i nf$, as the inference is reduced to a threshold check of the k_i values, thus leading to a negligible latency for a large fraction of input samples.

Indeed, it is sufficient to verify that the input of a neuron k_i remains under a certain threshold: $k_i < k^* - k^{(*,0)} \; \forall i$, a condition that must be valid for every neuron. This procedure can be effectively implemented on a microcontroller as it only requires comparison operations and depends on pre-computed parameters. Moreover, in the case of binary classification, there is no need to execute the inference at all.

Clearly, an asymmetric class can have a percentage of samples that lead to some activity. In this case, this reduced version introduces a drop in accuracy corresponding to the fraction of these samples (as reported in the experimental section). To reduce this drop, we have to increase the asymmetric threshold a^*. However, depending on the dataset and the task, it is not always possible to obtain the desired asymmetry.

In the case of anomaly detection, the asymmetric class will be the most represented one. In this case, the approach is viable as it is conservative: Some samples belonging to the "normal" class will be attributed to the "anomaly" class.

Anyway, we believe that it is possible to mitigate this drop with a less aggressive reduction. We will explore this in future works. However, this drop concerns only the reduced implementation, while the benefit of reducing spiking activity still remains from a computational and energy viewpoint.

5 Experimental results

In this section, we report experiments to evaluate the proposed eLSNN methodology in two different classification scenarios. The first scenario targets a multi-class classification task and focuses on the MNIST dataset [29]. As discussed in the introduction, this kind of task, while static, is commonly used to evaluate SNN performance. In our case, it is adopted to evaluate the spiking activity tuning in a general multi-class case using a public benchmark [24].

The second scenario focuses on SHM anomaly detection, a binary classification task which aims to predict the health status of a highway bridge. In this case, we use a benchmark resulting from a monitoring system consisting of an IoT node equipped with vibration sensors.

As the bridge underwent a repair intervention during monitoring, the dataset includes samples belonging to either the class "anomalous condition" before the intervention or "healthy condition" after the intervention. This dataset has been successfully used in various SHM state-of-the-art works.

Through this dataset, we evaluate the latency and energy consumption of eLSNN inference running on a microcontroller node and we show the impact of the spiking activity tuning methodology we propose in this work.

In Table 2, we provide an overview of state-of-art methods focusing on latency (timestep) reduction for comparison. In particular, we report the training algorithm, the dataset used and the timesteps for SNN computation. We also report the timesteps configurations used in this work, showing that the range of considered timesteps is in line with the state-of-the-art methods, but we are also able to reduce to zero timesteps for strong asymmetric network models. We remark that our method does not focus primarily on timestep reduction. Rather, we focus on spiking activity tuning. By reducing spiking activity for some classes, we can also reduce the required timesteps as a side effect. Indeed, in the extreme case in which we reach zero spikes for one specific class, we can perform the classification in zero timesteps. It is worth noting that recent work demonstrates how reducing the firing activity is more beneficial than minimizing timesteps in terms of energy efficiency and overall latency [45, 46].

5.1 Multi-class classification scenario

We applied the spiking activity tuning methodology proposed in Sect. 3.2 to the MNIST dataset [47]. The dataset is composed of 60,000 samples in the training set and 10,000 samples in the test set. The dataset is composed of 10 classes, one for each digit. Each sample is fed to the network without being pre-processed. The pixels are currents that will remain constant throughout the inference time (t_{inf}). From the original training set, we extracted 10,000 samples to be used as validation set and left the remaining 50,000 samples to train our models. Training, validation and test sets are balanced. In this scenario, the dataset used to train all the models ($\{\mathcal{M}\}$) has not been artificially unbalanced. This experiment evaluates the impact of hyper-parameters on spiking activity without modifying the public dataset. As such, Step 1 in Sect. 3.2 is not applied.[1]

The hyper-parameters explored in the grid search (Step 2 in Sect. 3.2) are the follows:

1. eLSNN hyper-parameters: (N = [100, 200, 300, 400], I = [768], t_{inf} = [5, 10, 20, 50] ticks, f^* = [0.1, 0.01, 0.001, 0]).
2. Other hyper-parameters: (τ_m = [20], τ_o = [3], v_{thc} = [0.01], β_c = [1.8], τ_{ac} = [0.5, 0.1], κ_p = [1, 100, 300], κ_f = [1, 100, 300])

From these models, we applied the policies defined in Step 3 in Sect. 3.2 to (i) select an optimal modeled (ii) evaluate the hyper-parameter impact on the eLSNN performance.

In Step 3, we select models with accuracy above 90%. On these models ($\mathcal{M}^*$), we first evaluated the impact of f^* (training parameter able to control the firing activity) on the different classes in the training set. The results of this test are depicted in Fig. 4. On the x-axis are reported for the different classes (highlighted with different colours) a box plot showing the average firing activity for the sample for the selected models. From the plot, we notice that on average the firing activity of the models computed in the test set decreases as the f^* decreases. Moreover, also the variability of the firing activities among the models trained with the same (f^*) decreases as well.

We observe that spiking activity reduction can be up to 2 orders of magnitude, which translates into a computational load reduction as discussed in Sect. 3. We then selected an optimal model ($\mathcal{M}^*$) applying an asymmetric behaviour threshold (a^*) equal to 90% in the Step 3. Looking at inference, Table 3 reports the performance for the selected optimal model computed on the test set.

The selected eLSNN model correctly classifies as class "1" the 93% of the samples with zero spikes (firing rates equal to zero) in the internal layer. For the remaining 7% (84 samples) of the samples, the network produces only on average 7 spikes. For the other classes, the average activity is about an order of magnitude higher. As discussed in

[1] Even if the unbalancing can be applied to a multi-class scenario, in this work, we limit our exploration of the unbalancing to the binary classification. Also, the unbalancing has a practical implication in the anomaly detection application, for which we devise an optimized implementation on a microcontroller node.

Table 2 Comparison of our work and the state of the art

Paper	Architecture	Training	Dataset	Timesteps
[17]	VGG9	SG	CIFAR10	20
	VGG11	SG	CIFAR100	30
	VGG11	SG	Tiny-ImageNet	25
	/	SG	DVS-CIFAR10	/
[18]	CNN	ANN-to-SNN	MNIST	19
	CNN	ANN-to-SNN	CIFAR10	204
	ResNet20	ANN-to-SNN	CIFAR11	198
	AE-CNN	ANN-to-SNN	CIFAR12	56
[34]	VGG16	IIR-SNN (Hybrid)	CIFAR10	1
	VGG16	IIR-SNN (Hybrid)	CIFAR100	1
	VGG16	IIR-SNN (Hybrid)	ImageNet	1
[35]	VGG16	SG	CIFAR10	2
	VGG17	SG	CIFAR100	2
[36]	VGG16	SG	CIFAR10	1
	VGG16	SG	ImageNet	1
[38]	VGG16	ANN-to-SNN	CIFAR10	8
	ResNet20	ANN-to-SNN	CIFAR10	8
	ResNet18	ANN-to-SNN	CIFAR10	8
[39]	VGG16	ANN-to-SNN	CIFAR100	1
	ResNet20	ANN-to-SNN	CIFAR100	1
	VGG16	ANN-to-SNN	ImageNet	1
	ResNet34	ANN-to-SNN	ImageNet	1
[40]	ResNet18	SG	CIFAR10	20
	ResNet18	SG	CIFAR100	20
	ResNet18	SG	ImageNet	50
	VGG11	SG	DVS-CIFAR10	20
[41]	VGG6	Hybrid	CIFAR10	5
	VGG16	Hybrid	CIFAR10	5
	ResNet20	Hybrid	CIFAR10	5
	VGG16	Hybrid	CIFAR100	5
	ResNet20	Hybrid	CIFAR100	5
	VGG16	Hybrid	ImageNet	5
[37]	VGG16	ANN-to-SNN	CIFAR10	3.17
	ResNet18	ANN-to-SNN	CIFAR10	2.52
	VGG16	ANN-to-SNN	ImageNet	3.26
	ResNet50	ANN-to-SNN	ImageNet	4.42
[42]	ResNet34	SG	ImageNet	2.40
	ResNet34	ANN-to-SNN	ImageNet	29.53
[43]	VGG9	SG	DVS-CIFAR10	5
	VGG9	SG	N-Caltech101	5
	VGG9	SG	DVS-GESTURE	5
Ours	**LSNN**	**SG**	**Anomaly Detection**	**0, 5, 10, 20**
	LSNN	**SG**	**MNIST**	**0, 5, 10, 20**

SG means surrogate gradient and hybrid means that the authors use mixed training, i.e. using both ANNs and SNNs

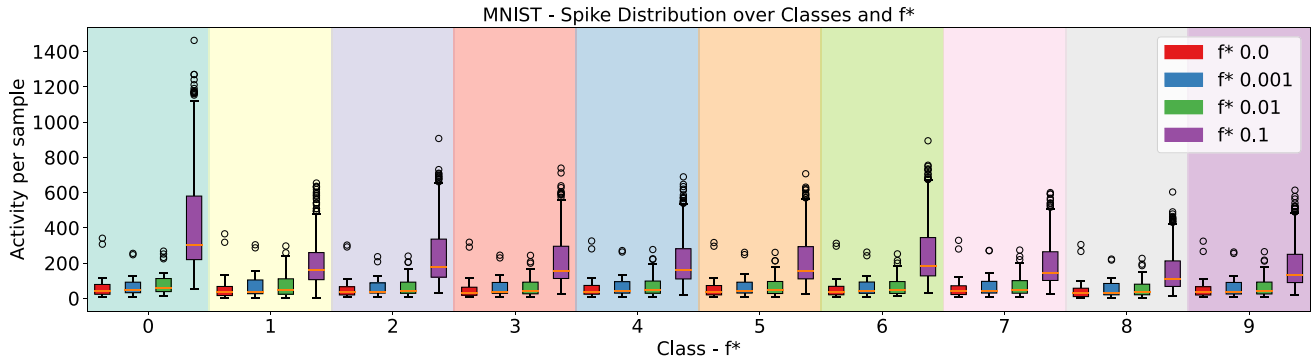

Fig. 4 Spike activity distribution of MNIST eLSNN configurations. We highlighted the mean activity per sample for each class (with different background colours) and for each configuration with a specific value of f^* (with different box-plot foreground colours)

Sect. 3, the proposed asymmetric eLSNNs can significantly reduce its computational cost for samples belonging to class "1" in the MNIST dataset.

We conclude that using the proposed methodology is effective in reducing spiking activity and that it is possible to impose an asymmetric behaviour. This characteristic will be exploited in the anomaly detection scenario, where we will show, additionally, that it is possible to impose the asymmetry for a specific class (i.e. the healthy class, which is more frequently seen at the network input).

5.2 Binary classification scenario

In this section, we apply the eLSNN tuning methodology described in Sect. 3.2 to an anomaly detection task of Structural Health Monitoring (SHM) [27]. The implemented pipeline is consistent with what is required in a real use case. It involves (i) sensor data acquisition and pre-processing, (ii) FFT computation and (iii) model inference. The pipeline has been executed on an IoT node (described later in this section) based on a STMicroelectronics microcontroller.

Table 3 Results of a eLSNN configuration with an asymmetric behaviour, trained with $f^* = 0$

Class	# Samples	# Samples ($z = 0$)	$\sum z$	a_i
0	995	0	60041	0.00
1	1233	1149	591	0.93
2	991	0	34255	0.00
3	957	0	25069	0.00
4	990	0	73179	0.00
5	840	0	55652	0.00
6	969	0	50213	0.00
7	1023	0	55824	0.00
8	981	0	29778	0.00
9	1021	0	45070	0.00

The table reports, for each class, the total number of samples (# Samples), the number of samples recognized with no spikes (# Samples ($z = 0$)), the total number of spikes ($\sum z$) and the asymmetry ratio a_i

The dataset is composed of a training set with 6394 samples, a validation set with 546 samples and a test set with 994 samples. All sets are balanced. Starting from the training set, we considered five artificially unbalanced versions of it obtained using a class sub-sampling (Step 1 in Sect. 3.2). For the SHM dataset (binary classification problem), we apply the following unbalancing ratios ($u^0_{\%} = 0.3, 0.4, 0.5, 0.6, 0.7$) where $u^0_{\%}$ represents the percentage of "Class 0" in the set. In this use case, we use a LSNN network composed of 20 ALIF and 2 output neurons which discriminate between two bridge states (healthy/damaged). The hyper-parameters explored in the grid search (Step 2 in Sect. 3.2) are:

(i) eLSNN hyper-parameters: (N = 20, I = [50, 150], t_{inf} = [5, 10, 20] ticks, f^* = [0.01, 0.001]) and
(ii) other hyper-parameters: (τ_m = [20, 30], τ_o = [3, 10, 30], v_{thc} = [0.01, 0.03], β_c = [1.7, 1.8], τ_{ac} [0.5, 1])

We applied the policies defined in Step 3 in Sect. 3.2 to select the best model among the trained ones and to evaluate how hyper-parameters impact eLSNN performance.

To characterize the runtime and energy impact of the eLSNN for the SHM use case, we used the following experimental setup: (i) A development board hosting an STM32F4 system-on-chip and (ii) a LabVIEW-controlled data acquisition board to measure the current absorbed by the MCU.

The SHM pipeline we developed ran on top of an STM32F4-Discovery board. It is a development platform produced by STMicroelectronics featuring the STM32F407VG6, a system-on-chip commonly used for DSP-oriented deeply embedded applications. Such a system hosts a Cortex-M4 core with a floating-point unit, supporting a maximum clock speed of 168 MHz, which

comes with 1 MiB Flash memory and 192 KiB of RAM. For the sake of our experiments, we configured the MCU as follows:

(i) The core is clocked at the maximum clock frequency of 168 MHz when running;
(ii) The MCU fetches structural accelerations from an embedded MEMS sensor via SPI;
(iii) The DMA is configured to transfer data while the core is in SLEEP mode, to reduce power consumption.

Current consumption is monitored by a PXIe-4309 board, produced by National Instruments. The PXIe-4309 module features 8 ADC devices and supports a maximum data acquisition rate of 2 MSamples per second. We used two of the available channels to:

(i) Sample the current absorbed by the MCU using a shunt resistor;
(ii) Read the logic level of a GPIO that the application software toggles each time a processing stage is started or completed, to associate each phase (e.g. SPI transfer, FFT computation, LSNN inference) to its current consumption.

The selection of the accurate models as described in Step 3 in Sect. 3.2 has been evaluated by looking at the Matthews Correlation Coefficient (MCC) on the validation set. We select in Step 3 the models with MCC higher than 0.75, which corresponds to an accuracy score over 88%. Even if this is a relatively low accuracy for MNIST classification using neural networks, we decided to use it for the sake of explanation of the trade-off between accuracy versus spiking activity.

On these models ($\mathcal{M}^*$), we first evaluated the impact of firing activity (z) and eLSNN hyper-parameters, namely $\#input$ (I) and inference time (t_{inf}), on the inference cost measured in cycles.

5.2.1 eLSNN hyper-parameters characterization

As described in Sect. 3, the eLSNN implementation leverages the sparse nature of the spikes for saving computations: If a neuron does not receive a spike, it does not trigger accumulation in the membrane potential. This means that the computational burden of the eLSNN algorithm depends on the average number of nonzero elements for each inference tick for the recursive/hidden layer. To understand the relevance of these effects on the total eLSNN inference time for the same flavour of eLSNN, we analysed different input samples (each corresponding to the FFT coefficients of the accelerations read by the MCU during a passage of a vehicle in the section of the bridge). More specifically, the number of spikes corresponding to nonzero elements in the recurrent layer during an eLSNN inference (denoted with z) depends on the specific input sample and network hyper-parameters. On the other side, differently from the recurrent/hidden layer activity (z), the spiking activity in the input layer depends only on the size of the input (I).

5.2.2 Computational cost of eLSNN

This section concludes the eLSNN study by comparing (on average on the test set) the performance of the optimal model selected with the proposed methodology asymmetric eLSNN against the best performing and energy-efficient eLSNN. This is obtained by evaluating the candidate eLSNNs in the median sample with respect to both x and z.

Indeed, from a separate characterization we performed (whose results are not reported here), we observed that the number of cycles strongly depends on the number of input neurons (I), and loosely on the spiking activity.

Figure 5 reports on the same plot the breakdown of the total number of cycles taken by the median inference time of both the eLSNN and asymmetric eLSNN. For the symmetric networks, all the eLSNN hyper-parameters combinations are reported, while for the asymmetric eLSNNs its performance does not change varying the inference ticks and thus we report only one network for I = 50 and I = 150. By comparing the different networks, we can notice that the two better performing eLSNNs (achieving MCC ≥ 0.75) are the configuration $I = 150, t_{inf} = 5$ and $I = 50, t_{inf} = 5$ for the eLSNN. As expected lowering t_{inf} reduces the computational cost. The asymmetric eLSNN always outperforms the eLSNN requiring lower cycles. We can conclude that asymmetric eLSNN is significantly more efficient than the eLSNN and that lowering the # of inputs (I) reduces the computational

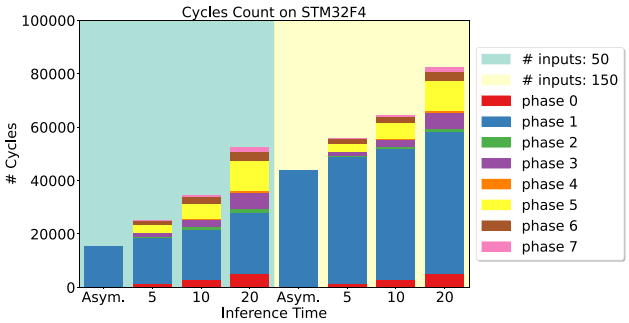

Fig. 5 Cycle count breakdown for eLSNN best configurations (input number-inference ticks) computed on samples with a median activity condition. The background colours identify the number of inputs, and the bar plot foreground colours the number of cycles for each eLSNN computation phase

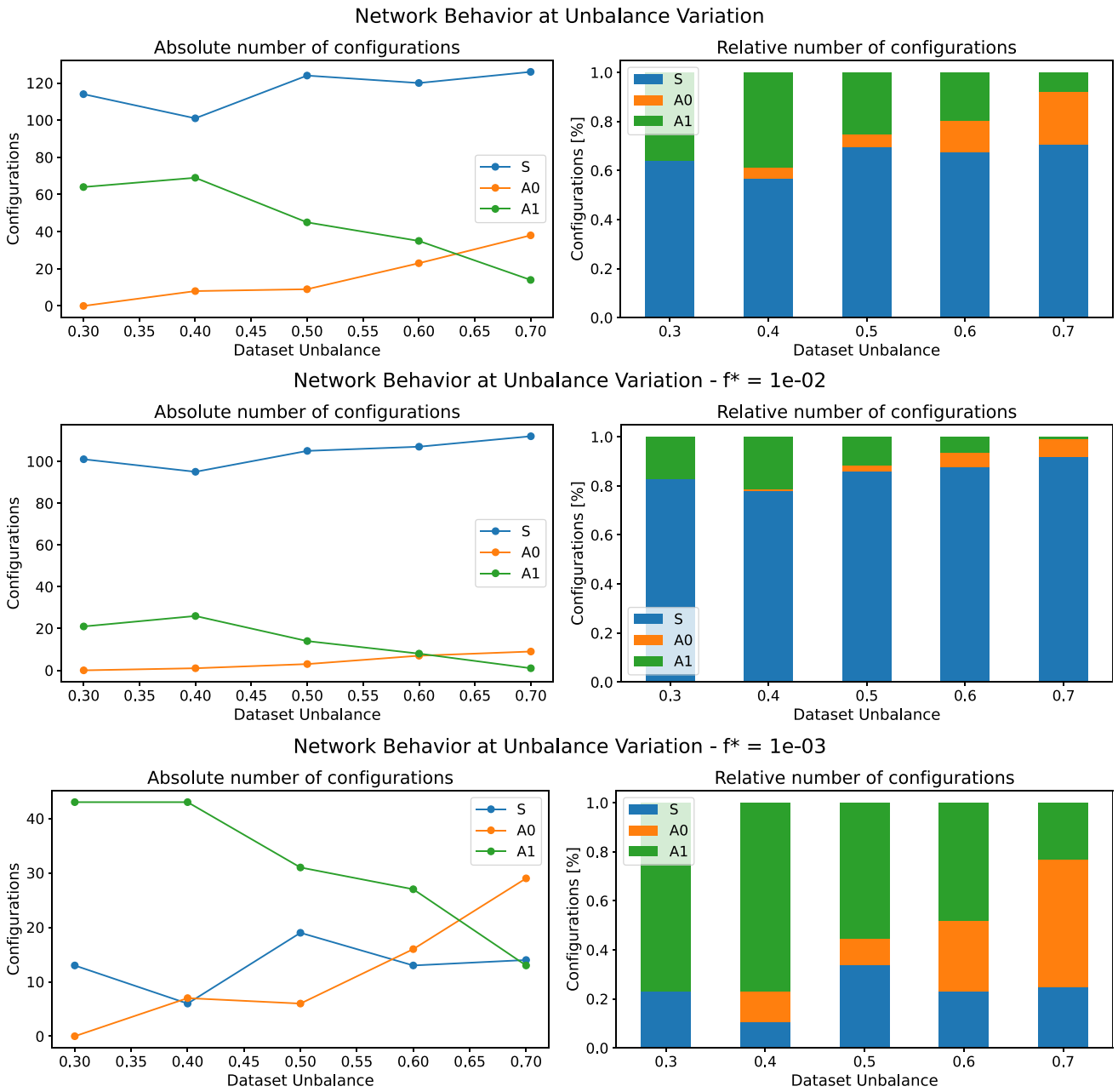

Fig. 6 In y-axis are reported the number (left plots) and percentage (right plots) of configurations with symmetric and asymmetric behaviour when class imbalance varies between 0.3 (more samples with class 1) and 0.7 (more samples with class 0) in the x-axis. Asymmetric networks are further divided in asymmetric configurations targeting class 0 (in orange) and asymmetric configurations targeting class 1 (in green)

load. Moreover, in Fig. 5, we report with different patterns the number of cycles needed to perform the computational steps described in Sect. 3. Phase 0: Compute of $\boldsymbol{v}^{(\alpha)}$ Phase 1: Compute of $\boldsymbol{v}^{(I)}$ Phase 2: Compute of $\boldsymbol{v}^{(H)}$ Phase 3: Compute of $\boldsymbol{a}^{(\xi)}$ Phase 4: Compute of $\boldsymbol{a}^{(th)}$ and $\boldsymbol{a}^{(z)}$ Phase 5: Compute of $\boldsymbol{z}$ and insert items in $\breve{\boldsymbol{z}}$ Phase 6: Compute of $\boldsymbol{y}$ Phase 7: Insert items in $\breve{\boldsymbol{x}}$ for next iteration. For the eLSNN, Phase 1 dominates the computational time since it computes the $\boldsymbol{v}^{in}$ vector using a matrix–vector multiplication. Phase 0 and Phase 3 involve N multiplications, and their contribution increases with the inference time t_{inf}.

5.2.3 Impact of asymmetric unbalance on training

In this subsection, we evaluate the impact of the dataset unbalance $u_{\%}$ and asymmetric threshold $a*$ in controlling the asymmetric behaviour of the eLSNN.

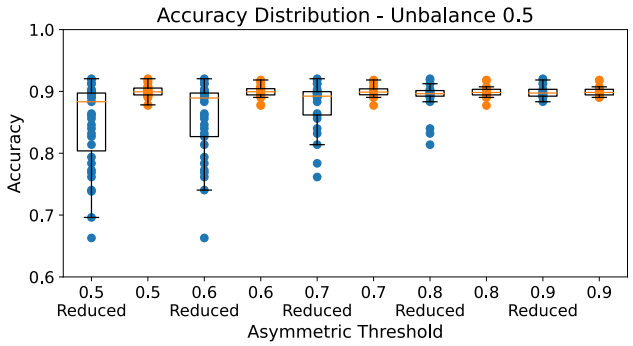

Fig. 7 Using an increasing asymmetry threshold, the accuracy of the configurations is evaluated via eLSNN in inference and via reduced eLSNN. Using the reduced eLSNN implementation for a model which is loosely asymmetric can lead to a degradation of the accuracy

Figure 6 reports the impact of the dataset unbalance for class "0" ($u^0_{\%}$) and of the f^* hyper-parameter on the asymmetric class learnt by the model. (The class which is classified by the model with zero spike activity.) In the figures, plots show with different colours the number of model configurations (absolute values on the right plot and relative impact on the left plots) which shows: (S) no asymmetric behaviour, $a_i < a^*$; (A0) asymmetric behaviour with the asymmetric class "0", $a_0 > a^*$; (A1) asymmetric behaviour with the asymmetric class "1", $a_1 > a^*$. On the x-axes, the dataset unbalances $u^0_{\%}$.

The top plot in Fig. 6 refers to all the models selected ($ACC > ACC*$). The central plot refers to the model trained with $f^* = 0.01$ while the bottom plot refers to the model trained with $f^* = 0.001$. From the figure, we can see that by decreasing the value of f^* the number of models trained with asymmetric behaviour increases, whereas the number of symmetric (S) models decreases. Approximately 20% of models have asymmetric behaviours with $f^* = 0.01$, and up to $\sim 90\%$ of models have asymmetric behaviours at $f^* = 0.001$.

Figure 6 also shows the impact of the dataset unbalance $u_{\%}$ on the control of the asymmetric class. Where the asymmetric class is defined as the class the network learns to represent with zero spikes in the internal layer. From the bottom plot, we can see that given a set of hyper-parameters the number of models which expose an asymmetric behaviour towards a given class is proportional to the representation of that given class in the training set. This is visible with the different bars as when the $u_{\%} = 0.3$ all the asymmetric models learn to represent the class "1" with zero firing rate. While increasing $u^0_{\%}$, the percentage of asymmetric models which learn to represent the class "0" with zero spikes, increases and reaches with $u^0_{\%} = 0.7$ the majority of the models.

We can explain the impact of dataset unbalance to the spiking activity by the fact that samples belonging to the same class likely create similar neuron activation patterns. During gradient computation these patterns strongly concur

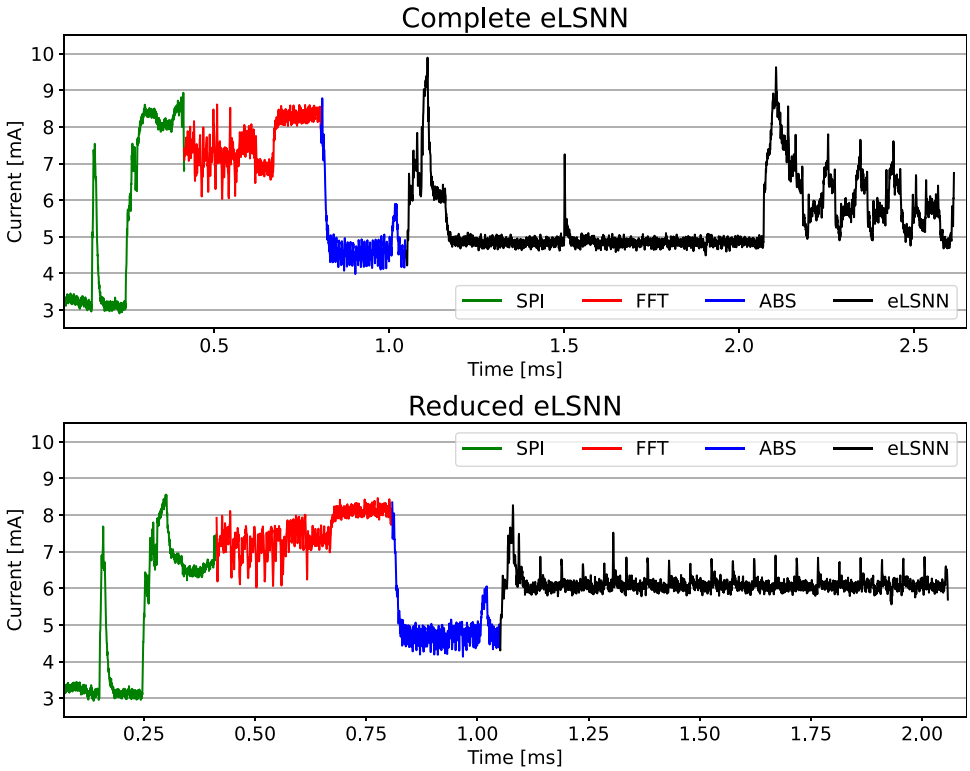

Fig. 8 SHM case study current traces for the complete (top) and reduced (bottom) eLSNN networks. For the sake of visualization, the traces were captured with the system-on-chip clock set to 16 MHz. The SPI master clock was configured to 8 MHz

Table 4 Report of the SHM case study featuring the complete eLSNNs tested

Stage	Complete eLSNN				
	Time [μs]	Cycles [#]	Current [mA]	Power [mW]	Energy [μJ]
SPI	541	–	6.03	17.08	9.78
FFT	41	6971	43.46	130.39	5.41
ABS	24	4032	41.80	125.40	3.01
eLSNN	148	24947	43.62	130.87	19.43
Overall	754	35950	33.73	100.94	37.63

Table 5 Report of the SHM case study featuring the reduced eLSNNs tested

Stage	Reduced (Asymmetric) eLSNN				
	Time [μs]	Cycles [#]	Current [mA]	Power [mW]	Energy [μJ]
SPI	545	–	5.72	17.17	9.36
FFT	41	6888	41.72	125.17	5.13
ABS	24	4032	40.75	122.24	2.93
eLSNN	95	16043	41.22	123.65	11.81
Overall	705	26963	32.35	97.06	29.23

to the loss (see Eq. 2), thus they are more impacted by the desired firing rate (f^*) parameter in the loss computation.

It is interesting to note that when the training dataset is balanced $u^0_{\%} = 0.5$ there is a majority of models which learn to represent class 1 with zero firing rates. This is unexpected as one would have supposed to have a balanced composition of asymmetric classes in the models. Future works will investigate this behaviour.

In Fig. 7 we report the accuracy computed for the test set using the eLSNN and the asymmetric eLSNN model (using the reduced implementation described in Sect. A) selected using a different asymmetric threshold (a^*). As expected, using the reduced eLSNN implementation for a model which is loosely asymmetric ($a^* < 0.9$) leads to a degradation of the accuracy. This is expected for a loosely asymmetric model, as, by construction, the number of samples leading to zero spiking activity is less. Because the reduced implementation bases the recognition on the assumption that samples of a given class lead to zero spikes, a sample belonging to this class but generating internal spikes will be misclassified.

As discussed at the end of Sect. A, to mitigate this drop it is possible to increase the asymmetric threshold a^*. Depending on the characteristics of the dataset and the task, it may not be possible to obtain a model satisfying this asymmetry requirement with the wanted performance. Exploring to which extent it is possible to apply this optimization to different datasets will be the purpose of future works.

In general, in the case of anomaly detection, the accuracy drops may be acceptable as it goes in the conservative direction: Some samples belonging to the "normal" class will be attributed to the "anomaly" class. Furthermore, this limitation impacts only the reduced implementation. For a not-reduced implementation (where the optimization presented in Sect. A is not adopted), the benefit of reducing spiking activity, by playing with asymmetry, still remains from a computational and energy viewpoint.

5.2.4 Energy and latency results on HW microcontroller node

Finally, we report the energy and latency results of an execution of eLSNN on the SHM microcontroller node. In particular, we used the eLSNN tuning methodology to select two optimal models ($\mathcal{M}^*_{1,2}$) applying (1) an asymmetric behaviour threshold (a^*) equal to 90% in the Step 3 (see Sect. 3.2) and (2) applying only the condition of $min(z)$ but without selecting an asymmetric behaviour.

We selected the latter as a reference to show the impact of the asymmetric models with respect to a simpler approach that tries to minimize the firing activity in a uniform way across the classes. Model (1) is then an asymmetric model, and we refer to this model as the reduced eLSNN. Model (2) instead presents a low-firing rate for both the classes but none of them learned with a zero firing rate, and we refer to this model as the complete eLSNN. Both the models have $I = 50$ and $t_{inf} = 5$. We used both these models in the entire SHM pipeline.

Figure 8 shows the MCU 's current consumption when executing the entire processing steps needed to read MEMS values, pre-process them and run eLSNN inference. On the top plot, we report the Complete eLSNN in its most efficient configuration, and on the bottom plot, we report the Reduced eLSNN in its most efficient configuration. With different colours, we depict the SHM pipeline phases: (i) Raw data fetch over SPI and conversion to real acceleration values (SPI), (ii) FFT computation (FFT), (iii) computation of FFT coefficients absolute values (ABS) and finally (iv) eLSNN inference (eLSNN). We notice that the proposed reduced (asymmetric) eLSNN is capable of reducing the execution time of the entire pipeline significantly from 2.5 to 2 ms (with the microcontroller configured at 16 MHz.)

Tables 4 and 5 summarize all these effects. We can notice that the reduced eLSNN kernel takes 1.56× fewer

cycles than the Complete eLSNN network. When the total computation time is considered this speed up reduces to 1.33×. Moreover, if we consider the energy consumption, the reduced eLSNN (Asymmetric) reduces the energy consumption by 1.29×.

We can conclude that the proposed methodology for tuning LSNN models combined with the efficient eLSNN implementation can achieve significant savings while executing on resource-constrained hardware.

6 Conclusion

In this paper, we presented a methodology to tune the spiking activity of LSNNs to reduce their execution cycles and energy consumption during a classification task. We reported the analysis we conducted to extract the network and dataset hyper-parameters impacting neuron firing and we measured the effect of their tuning on the spiking activity occurring when samples belonging to the various classes are presented to the network during inference. We reported such an impact in the case of binary and multi-class classification scenarios. Tuning can lead to 68–85% activity reduction with respect to the average value depending on the class.

We described an optimized implementation (eLSNN) running on a low-power microcontroller and we demonstrate the effectiveness of the tuning strategy when processing a real dataset for a SHM application, where the objective is the classification of a viaduct as damaged or healthy. We performed a characterization of the computational cycles required for the inference.

We also show that in the extreme case where one of the classes reaches zero spiking activity, it is possible to further optimize the network implementation. Clearly, this is possible depending on the complexity of the task. However, our method shows that this condition can be found through the proposed spiking activity tuning method.

We exploited an experimental setup to gather real energy measurements featuring a hardware-in-the-loop approach to inject real data into the sensor node.

Results show 25% cycles count and 22% energy reduction when comparing the reduced versus a complete eLSNN version. Even though it has been evaluated on a digital microcontroller, the proposed methodology is applicable also to dedicated hardware architectures, provided a suitable training algorithm is available. Future work will be devoted to applying the proposed technique to neuromorphic hardware and to applying the tuning procedure to quantized networks.

Appendix A: Derivation of the formula for the evaluation of the network output

According to Eq. (5), in the presence of a constant input current each neuron in the internal layer receives a constant stimulus equal to $k_i = \sum_j W_{ji}^{in} x_j^t$. Considering initially null neuronal activity ($z_i^{t=0} = 0 \forall i$), it is possible to express the evolution of the membrane voltage by the following equation:

$$\begin{aligned} v_i^t &= \alpha^t v_0 + k_i \sum_{\tau=0}^{t-1} \alpha^\tau \\ &= \alpha^t v_0 + k_i \frac{1-\alpha^t}{1-\alpha} \end{aligned} \tag{A1}$$

Using v_i^t in the absence of internal activity and with constant input, it is possible to identify the time of the first spike for each neuron (considering $v_i^t = v_{th}$):

$$\begin{aligned} &t = log_\alpha \left(\frac{v_{th}(1-\alpha) - k_i}{v_0(1-\alpha) - k_i} \right) \\ &\textbf{if } v_0 = 0 \longrightarrow t = log_\alpha \left(1 - \frac{v_{th}(1-\alpha)}{k_i} \right) \end{aligned} \tag{A2}$$

If the inference time of an eLSNN is known and is equal to T it is possible to identify the range of values k_i that given as input to a neuron do not cause the emission of a spike:

$$\begin{aligned} v_i^T &< v_{th} \\ \alpha^T v_0 + k_i \frac{1-\alpha^T}{1-\alpha} &< v_{th} \\ k_i &< \underbrace{v_{th} \frac{1-\alpha}{1-\alpha^T}}_{k^*} - \underbrace{v_0 \alpha^T \frac{1-\alpha}{1-\alpha^T}}_{k^{(*,0)}} \end{aligned} \tag{A3}$$

Funding Open access funding provided by Alma Mater Studiorum - Università di Bologna within the CRUI-CARE Agreement.

Data availability The SHM dataset used in the current study is available from the corresponding author on reasonable request. Restrictions apply to the availability of these data, which were used under licence for the current study, and so are not publicly available.

Declarations

Conflict of interest The authors of this paper have declared that they do not have any conflict of interest to disclose.

References

1. Maass W (1997) Networks of spiking neurons: the third generation of neural network models. Neural Netw 10(9):1659–1671. https://doi.org/10.1016/S0893-6080(97)00011-7
2. Liu J, Lu H, Luo Y, Yang S (2021) Spiking neural network-based multi-task autonomous learning for mobile robots. Eng Appl Artif Intell 104:104362. https://doi.org/10.1016/j.engappai.2021.104362
3. Barchi F, Zanatta L, Parisi E, Burrello A, Brunelli D, Bartolini A, Acquaviva A (2021) Spiking neural network-based near-sensor computing for damage detection in structural health monitoring. Fut Internet. https://doi.org/10.3390/fi13080219
4. Indiveri G, Horiuchi T (2011) Frontiers in neuromorphic engineering. Front Neurosci 5:118. https://doi.org/10.3389/fnins.2011.00118
5. Roy K, Jaiswal A, Panda P (2019) Towards spike-based machine intelligence with neuromorphic computing. Nature 575(7784):607–617. https://doi.org/10.1038/s41586-019-1677-2
6. Deng L, Wu Y, Hu X, Liang L, Ding Y, Li G, Zhao G, Li P, Xie Y (2020) Rethinking the performance comparison between snns and anns. Neural Netw 121:294–307. https://doi.org/10.1016/j.neunet.2019.09.005
7. Lamba S, Lamba R (2019) Spiking neural networks vs convolutional neural networks for supervised learning. In: 2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), pp 15–19. https://doi.org/10.1109/icccis48478.2019.8974507
8. Wu X, Saxena V, Zhu K (2015) Homogeneous spiking neuromorphic system for real-world pattern recognition. IEEE J Emerg Select Top Circ Syst. https://doi.org/10.1109/jetcas.2015.2433552
9. Sakai Y, Pedroni BU, Joshi S, Tanabe S, Akinin A, Cauwenberghs G (2019) Dropout and dropconnect for reliable neuromorphic inference under communication constraints in network connectivity. IEEE J Emerg Select Top Circ Syst 9(4):658–667. https://doi.org/10.1109/jetcas.2019.2952642
10. Stanojevic A, Cherubini G, Woźniak S, Eleftheriou E (2023) Time-encoded multiplication-free spiking neural networks: application to data classification tasks. Neural Comput Appl 35(9):7017–7033. https://doi.org/10.1007/s00521-022-07910-1
11. Madrenas J, Zapata M, Fernández D, Sánchez-Chiva JM, Valle J, Mata-Hernández D, Oltra JA, Cosp-Vilella J, Sato S (2020) Towards efficient and adaptive cyber physical spiking neural integrated systems. In: 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp 1–4. https://doi.org/10.1109/icecs49266.2020.9294982
12. Zhao J, Risi N, Monforte M, Bartolozzi C, Indiveri G, Donati E (2020) Closed-loop spiking control on a neuromorphic processor implemented on the icub. IEEE J Emerg Select Top Circ Syst 10(4):546–556 https://doi.org/10.1109/jetcas.2020.3040390. arXiv:2009.09081 [cs.ET]
13. Eshraghian JK, Ward M, Neftci E, Wang X, Lenz G, Dwivedi G, Bennamoun M, Jeong DS, Lu WD (2023) Training spiking neural networks using lessons from deep learning. Proceedings of the IEEE 111(9):1016–1054. https://doi.org/10.1109/JPROC.2023.3308088
14. Bellec G, Scherr F, Subramoney A, Hajek E, Salaj D, Legenstein R, Maass W (2020) A solution to the learning dilemma for recurrent networks of spiking neurons. Nat Commun 11:1–15
15. Lee JH, Delbruck T, Pfeiffer M (2016) Training deep spiking neural networks using backpropagation. Front Neurosci. https://doi.org/10.3389/fnins.2016.00508
16. Valencia D, Alimohammad A (2023) A generalized hardware architecture for real-time spiking neural networks. Neural Comput Appl. https://doi.org/10.1007/s00521-023-08650-6
17. Kim Y, Panda P (2021) Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. Front Neurosci. https://doi.org/10.3389/fnins.2021.773954
18. Hwang S, Chang J, Oh M-H, Min KK, Jang T, Park K, Yu J, Lee J-H, Park B-G (2021) Low-latency spiking neural networks using pre-charged membrane potential and delayed evaluation. Front Neurosci. https://doi.org/10.3389/fnins.2021.629000
19. Moradi S, Qiao N, Stefanini F, Indiveri G (2017) A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). IEEE Trans Biomed Circuits Syst 12(1):106–122
20. Schemmel J, Fieres J, Meier K (2008) Wafer-scale integration of analog neural networks. In: 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), pp 431–438. Ieee
21. Benjamin BV, Gao P, McQuinn E, Choudhary S, Chandrasekaran AR, Bussat J, Alvarez-Icaza R, Arthur JV, Merolla PA, Boahen K (2014) Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. Proc IEEE 102(5):699–716
22. Davies M, Srinivasa N, Lin T-H, Chinya G, Cao Y, Choday SH, Dimou G, Joshi P, Imam N, Jain S et al (2018) Loihi: a neuromorphic manycore processor with on-chip learning. IEEE Micro 38(1):82–99
23. ...DeBole MV, Taba B, Amir A, Akopyan F, Andreopoulos A, Risk WP, Kusnitz J, Otero CO, Nayak TK, Appuswamy R, Carlson PJ, Cassidy AS, Datta P, Esser SK, Garreau GJ, Holland KL, Lekuch S, Mastro M, McKinstry J, Nolfo C, Paulovicks B, Sawada J, Schleupen K, Shaw BG, Klamo JL, Flickner MD, Arthur JV, Modha DS (2019) Truenorth: accelerating from zero to 64 million neurons in 10 years. Computer 52(05):20–29. https://doi.org/10.1109/mc.2019.2903009
24. He W, Wu Y, Deng L, Li G, Wang H, Tian Y, Ding W, Wang W, Xie Y (2020) Comparing snns and rnns on neuromorphic vision datasets: similarities and differences. Neural Netw. https://doi.org/10.1016/j.neunet.2020.08.001
25. Datta G, Kundu S, Jaiswal AR, Beerel PA (2022) Ace-snn: algorithm-hardware co-design of energy-efficient & low-latency deep spiking neural networks for 3d image recognition. Front Neurosci. https://doi.org/10.3389/fnins.2022.815258
26. Liu J, Lu H, Luo Y, Yang S (2021) Spiking neural network-based multi-task autonomous learning for mobile robots. Eng Appl Artif Intell 104:104362. https://doi.org/10.1016/j.engappai.2021.104362
27. Burrello A, Marchioni A, Brunelli D, Benatti S, Mangia M, Benini L (2020) Embedded streaming principal components analysis for network load reduction in structural health monitoring. IEEE Internet Things J 8(6):4433–4447. https://doi.org/10.1109/JIOT.2020.3027102
28. Pang L, Liu J, Harkin J, Martin G, McElholm M, Javed A, McDaid L (2020) Case study-spiking neural network hardware system for structural health monitoring. Sensors. https://doi.org/10.3390/s20185126
29. LeCun Y, Cortes C, Burges C (2010) MNIST handwritten digit database. Florham Park, NJ, USA

30. Häusser M (2000) The hodgkin-huxley theory of the action potential. Nat Neurosci 3(11):1165–1165. https://doi.org/10.1038/81426
31. Izhikevich EM (2003) Simple model of spiking neurons. IEEE Trans Neural Netw 14(6):1569–1572. https://doi.org/10.1109/tnn.2003.820440
32. Burkitt AN (2006) A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. Biol Cybern 95(1):1–19. https://doi.org/10.1007/s00422-006-0068-6
33. Werbos PJ (1990) Backpropagation through time: what it does and how to do it. Proc IEEE 78(10):1550–1560
34. Chowdhury SS, Rathi N, Roy K (2021) One timestep is all you need: training spiking neural networks with ultra low latency. https://doi.org/10.48550/arXiv.2110.0592
35. Datta G, Beerel PA (2022) Can deep neural networks be converted to ultra low-latency spiking neural networks? In: Proceedings of the 2022 Conference & Exhibition on Design, Automation & Test in Europe. Date '22, pp 718–723. European Design and Automation Association, Leuven, BEL
36. Datta G, Liu Z, Beerel PA (2022) Hoyer regularizer is all you need for ultra low-latency spiking neural networks. https://doi.org/10.48550/arXiv.2212.10170
37. Li C, Jones EG, Furber S (2023) Unleashing the potential of spiking neural networks with dynamic confidence. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp 13350–13360
38. Bu T, Ding J, Yu Z, Huang T (2022) Optimized potential initialization for low-latency spiking neural networks. AAAI Conf Artif Intell 36(1):11–20. https://doi.org/10.1609/aaai.v36i1.19874
39. Hao Z, Ding J, Bu T, Huang T, Yu Z (2023) Bridging the Gap between ANNs and SNNs by Calibrating Offset Spikes. ICLR 2023
40. Meng Q, Xiao M, Yan S, Wang Y, Lin Z, Luo Z (2022) Training high-performance low-latency spiking neural networks by differentiation on spike representation. In: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp 12434–12443. IEEE Computer Society, Los Alamitos, CA, USA. https://doi.org/10.1109/cvpr52688.2022.01212 . https://doi.ieeecomputersociety.org/10.1109/CVPR52688.2022.01212
41. Rathi N, Roy K (2023) Diet-snn: a low-latency spiking neural network with direct input encoding and leakage and threshold optimization. IEEE Transs Neural Netw Learn Syst 34(6):3174–3182. https://doi.org/10.1109/tnnls.2021.3111897
42. Li Y, Geller T, Kim Y, Panda P (2024) Seenn: towards temporal spiking early exit neural networks. In: Proceedings of the 37th International Conference on Neural Information Processing Systems (NIPS '23). Curran Associates Inc., Red Hook, NY, USA, Article 2764, 63327–63342
43. Ding Y, Zuo L, Jing M, He P, Xiao Y (2024) Shrinking your timestep: towards low-latency neuromorphic object recognition with spiking neural network. arXiv preprint arXiv:2401.01912
44. Chowdhury SS, Rathi N, Roy K (2022) Towards ultra low latency spiking neural networks for vision and sequential tasks using temporal pruning. In: Avidan S, Brostow G, Cissé M, Farinella GM, Hassner T (eds) Computer Vision - ECCV 2022. Springer, Cham, pp 709–726
45. Zanatta L, Di Mauro A, Barchi F, Bartolini A, Benini L, Acquaviva A (2023) Directly-trained spiking neural networks for deep reinforcement learning: energy efficient implementation of event-based obstacle avoidance on a neuromorphic accelerator. Neurocomputing 562:126885
46. Di Mauro A, Prasad AS, Huang Z, Spallanzani M, Conti F, Benini L (2022) Sne: an energy-proportional digital accelerator for sparse event-based convolutions. In: 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 825–830. IEEE
47. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324