Neurocomputing 562 (2023) 126885

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Directly-trained Spiking Neural Networks for Deep Reinforcement Learning: Energy efficient implementation of event-based obstacle avoidance on a neuromorphic accelerator

Luca Zanatta [a,*], Alfio Di Mauro [b], Francesco Barchi [a], Andrea Bartolini [a], Luca Benini [a,b], Andrea Acquaviva [a]

[a] DEI Department, University of Bologna, Bologna,
[b] Institut für Integrierte Systeme, ETH, Zürich, Switzerland

ARTICLE INFO

Communicated by F. Perez-Pena

Keywords:
Spiking Neural Networks
Neuromorphic computing
Neuromorphic hardware
Reinforcement learning
DQN
UAV

ABSTRACT

Spiking Neural Networks (SNN) promise extremely low-power and low-latency inference on neuromorphic hardware. Recent studies demonstrate the competitive performance of SNNs compared with Artificial Neural Networks (ANN) in conventional classification tasks.

In this work, we present an energy-efficient implementation of a Reinforcement Learning (RL) algorithm using SNNs to solve an obstacle avoidance task performed by an Unmanned Aerial Vehicle (UAV), taking a Dynamic Vision Sensor (DVS) as event-based input. We train the SNN directly, improving upon state-of-art implementations based on hybrid (not directly trained) SNNs. For this purpose, we devise an adaptation of the Spatio-Temporal Backpropagation algorithm (STBP) for RL. We then compare the SNN with a state-of-art Convolutional Neural Network (CNN) designed to solve the same task. To this aim, we train both networks by exploiting a photorealistic training pipeline based on AirSim. To achieve a realistic latency and throughput assessment for embedded deployment, we designed and trained three different embedded SNN versions to be executed on state-of-art neuromorphic hardware, targeting state-of-the-art.

We compared SNN and CNN in terms of obstacle avoidance performance showing that the SNN algorithm achieves better results than the CNN with a factor of 6× less energy. We also characterize the different SNN hardware implementations in terms of energy and spiking activity.

Spiking Neural Networks (SNN) are sometimes referred to as the third generation of Artificial Neural Networks (ANN) [1] since their behaviour is closer to the mammalian biological brain than the previous generations. The main difference between the standard neural networks, also referred to as the second generation ANNs, and the SNNs is the computational unit, namely the neuron. In ANNs, neurons are described by a function called activation function, while in SNNs, neurons, called spiking neurons, are modelled by a system of Ordinary Differential Equations (ODE). An ODE system describes the spiking neuron as a dynamic system in which a hidden state, called membrane potential, is accumulated. If it crosses a threshold on the rising edge, it emits an event (or pulse), also referred to as a spike. Spikes are the means for neurons to communicate with each other; a sequence of them is called a spike train. The presence of the spikes enables asynchronous communication among the neurons. Due to the complex nature of the SNNs, they hold the potential to have the potential to mimic key features of the mammalian brain, such as low latency, fast inference and energy efficiency.

The scientific community and major semiconductor companies, like Intel and IBM, have recently proposed optimized hardware implementation of the spiking neurons and/or the asynchronous communication between them, with the goal of (i) simulating neurocircuitries efficiently, (ii) evaluating the computational efficiency of SNNs on lab experiments, and (iii) apply SNNs on real-life problems.

The hardware platforms that nowadays support the direct execution of SNNs on specialized hardware are often referred to as "neuromorphic". Given the multitude of goals and youth of the technology, several conceptually different silicon implementations do exist. Intel Loihi [2] is a digital chip developed by Intel, consisting of an asynchronous mesh connecting programmable digital neurons. DYNAPs is a mixed-signal chip in which the neurons are described by analogue circuits connected by a digital asynchronous network [3]. TrueNorth is a digital chip developed by IBM [4] and consists of digital neurosynaptic cores and an asynchronous network-on-chip. Larger systems, which allow

* Corresponding author.
E-mail address: luca.zanatta3@unibo.it (L. Zanatta).

https://doi.org/10.1016/j.neucom.2023.126885
Received 17 October 2022; Received in revised form 18 February 2023; Accepted 3 October 2023
Available online 7 October 2023

simulating large-scale SNNs in real time, have also been proposed in the past, for example, Neurogrid [5]. In the embedded domain, SNE is a digital IP targeting neuromorphic accelerators to be integrated into heterogeneous SoC. Indeed, SNE provides ultra-low power SNN inference thanks to its custom and specialized microarchitecture. SNE has been integrated with tightly coupled standard RV32 host processors and multi-core cluster accelerator in the Kraken SoC [6].

Thanks to that, SNE can be reprogrammed at a low cost by the RV32 host processor. Furthermore, spiking events can be exchanged between SNE and the host processor.

Different families of training algorithms have been proposed for SNNs: (i) the biological-inspired ones, which are based on the Spike-Timing-Dependent-Plasticity (STDP); (ii) hybrid methods which convert the trained ANNs into SNNs and do not use the full expressiveness of the SNN in the training phase and (iii) gradient-based methods which approximate the spike derivative.

Deep Reinforcement Learning (RL) is helpful for training robots since it is challenging to create an offline dataset that catches all the features of the environment, while the RL paradigm allows the robot to learn in real time from direct interaction with the environment [7]. Furthermore, RL has been successfully exploited to train a robot in a simulation environment before deploying it in the real world [8].

RL tasks are characterized by a temporal dimension and high complexity. In this context, SNNs represent a promising approach to solving such tasks efficiently [9]: on one side, they inherit a time dimension feature, and as neuron dynamics capable of storing information in the membrane potential, on the other side, it has been proven that SNNs have a higher expressiveness compared to conventional ANNs [1]. Until now, RL tasks have been typically deployed on highly computationally capable platforms [10]. This trend is changing, and complex cognitive tasks are expected to be deployed on more computational resource-constrained nano, and pico-sized vehicles [10]. In this domain, the energy efficiency of SNNs is particularly appealing.

Since deep reinforcement learning with SNNs is less studied compared to supervised learning, it is essential to compare SNNs with the more established ANNs to establish a solid baseline. In this paper, we compare an SNN and an ANN in a challenging obstacle avoidance task in which the networks are trained with the RL algorithm. More in detail, the main contributions of the paper are:

- An adaptation of the Spatio-Temporal BackPropagation (STBP) [11] SNN training method for RL.
- A direct comparison between an ANN and its SNN counterpart, trained on the same RL setting.
- The creation of a plugin for QuantLab [12] to support the quantization of the proposed SNN.
- A design and implementation of the SNN on low-power neuromorphic hardware, the SNE [13].
- A new approach, called pqeSCNN, leverages the capability of SNE to be embedded in an SoC to reduce the accuracy drop induced by the HW quantization. The proposed technique computes the last layer of the SNN at full FP32 accuracy by the host processor, while the remaining layers are computed with 4 bit quantization by the SNE HW accelerators.
- An evaluation of the energy consumption of the proposed embedded SNN when deployed on a dedicated hardware accelerator.

To train and evaluate the neural networks, we create an end-to-end photorealistic pipeline in which the environment is a 70 m long lane full of obstacles. In this setup, we show that the SNN has a higher performance since it reaches the end of the lane 98% of the time, compared to 89% of the CNN.

Furthermore, we adapt the SNN to fit the SNE (the neuromorphic accelerator integrated into the Kraken SoC) computation engine characteristics, particularly the 4 bit quantized weights and the 8 bit quantized membrane potential and the absence of biases. To this aim, we first explored the performance of the SNN by removing biases: This network reaches the goal 77% of the time. By fixing as a reference the performance of the original SNN, we obtain that the distance that 98% of the drones can reach is 35 m.

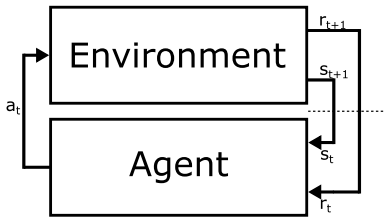


Fig. 1. Reinforcement training loop. The agent acts in the environment thanks to the action a_t, the environment evolves and communicates the new state s_t and the reward r_t to the agent.

Subsequently, we designed two quantized networks: (i) One (qeSCNN), which accumulates the membrane potential in 8 bit integer variables that can entirely run in SNE; (ii) The other (pqeSCNN), which computes the last layer membrane potential in the Kraken RISC-V-based multi-core cluster using 32 bit floating-point variables. Both networks are post-training quantized. These two networks allow us to evaluate the impact of post-training quantization on the SNN. This turns out to be more severe than the bias removal, as both the two networks do not reach the goal. Using the original SNN as a reference (distance reached by 98% of drones), the first network (qeSCNN) achieves a distance of 1 m while the latter reaches 16 m.

Regarding energy consumption, we report the inference energy measurements of the quantized versions, compared with an implementation of the CNN version on an SoA reference architecture. Notwithstanding the remaining performance drop of the quantized versions, we conclude that SNN can perform the considered RL task with an order of magnitude less energy than CNNs, opening the way to further exploration of their adoption in RL.

1. Background

1.1. Reinforcement learning

The Reinforcement Learning (RL) paradigm is a hot area of research since it promises to solve complex problems in control tasks and robotics and because it mimics the human learning behaviour too [7]. RL differs from supervised and unsupervised learning since it has the following distinctive characteristics: (i) a lack of oracle, (ii) sparsity of feedback and (iii) data generated during the training [7]. Fig. 1 shows a generic RL loop. The agent acts in the environment and, as a consequence of its actions, the environment changes. Examples of agents are drones, cars, robots, humans, neural networks, etc. The environment is what surrounds the agent. In RL, simulators are used to mimic an agent (or more) and its environment. As it is shown in Fig. 1, the agent, according to an action-producing function called policy (π), chooses an action (a_t) for modifying the environment. When the action is chosen, the environment will change according to the transition function. The environment communicates the goodness of the action chosen through rewards (r_t) and the new state (s_t) reached by the agent. Since the sensors used by the agent for tracking the state may not detect all the features of the environment, a history of a few stacked samples S is used [7].

An RL algorithm can be classified as follows:

- model-based or model-free;
- policy-based or value-based;
- on-policy or off-policy.

A model-based algorithm is a powerful model in which the algorithm tries to predict the following observations and/or rewards, acting in the best possible way to achieve the goal. The model-based models are used when the environment and its rules are well known. An example is the RL applied to chess games [14]. On the opposite side, there are model-free algorithms. In this set, the algorithm does not build the model of the environment. This family is suitable when the environment and its rules are not well known or when the environment is not deterministic [14]. A policy-based algorithm is an algorithm in which the policy is learned, i.e. the agent learns the distribution of the actions that the agent should perform at each step. In contrast, a value-based algorithm is an algorithm in which the agent learns to give a score to each action and then acts in the environment with the best one. The distinction between on-policy and off-policy regards the use of the samples collected by the agent: if the training of the agent is performed with the samples collected from different policies, the algorithm is called off-policy otherwise is called on-policy [14]. In our work, we use a Double Deep Q-Network (D2QN), which is a model-free, value-based and off-policy algorithm.

1.2. Spiking Neural Networks

A Spiking Neural Network (SNN) is a network that is composed of spiking neurons. In literature, various neuron models have been proposed, and one of the most common is the Leaky Integrate-and-Fire (LIF) model, described by the following equation:

$$\tau \frac{dv(t)}{dt} = -v(t) + I(t) \tag{1}$$

where τ is a time constant, $v(t)$ is the membrane potential and $I(t)$ is the pre-synaptic current. For the sake of discrete time algorithmic implementation, Eq. (1) can be rewritten recurrently, in which the membrane potential at time $t+1$ is a function of the membrane potential at time t:

$$v(t+1) = v(t)e^{-\frac{dt}{\tau}} + I(t) \tag{2}$$

where dt is the integration step of the simulation of the network. We will refer to it as *tick*. The input current is defined as:

$$I_i^n(t) = b_i^n + \sum_{j=1}^{l(n-1)} w_{ij}^n z_j^{n-1}(t-1) \tag{3}$$

where n is the layer, j is the index of the pre-synaptic neuron, i is the index of the post-synaptic one, b is the bias (a.k.a. constant current), w are the weights, and $z(t)$ are the impulses emitted by the neurons, called spikes:

$$z(t+1) = \begin{cases} 1 & \text{if } v(t) \geq v_{th} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

According to Eq. (4), a spike is generated if the membrane potential reaches the threshold in its rising edge.

Depending on the design, the neuron, after emitting a spike, can evolve in two different ways: (i) Reset its $v(t)$ to a fixed value called rest membrane potential v_{rest} (hard-reset) [11]:

$$v(t+1) = \begin{cases} v_{rest} & \text{if } z(t) = 1 \\ v(t+1) & \text{if } z(t) = 0 \end{cases} \tag{5}$$

(ii) subtracting the threshold from the $v(t)$ (soft-reset) [9]:

$$v(t+1) = \begin{cases} v(t) - v_{thr} & \text{if } z(t) = 1 \\ v(t+1) & \text{if } z(t) = 0 \end{cases} \tag{6}$$

If the spiking behaviour is disabled, the neuron is a "non-spiking LIF". In our work, we use both neuron models.

1.3. Spiking Neural Network training

The training of SNNs is a non-trivial task since the function used to create the spikes is not differentiable. Hence different training methods have been developed. We can group these methods into (i) Biological accurate methods, (ii) hybrid methods and (iii) gradient descend-based methods. In the first family, we find all the methods based on Spike-Timing-Dependent Plasticity (STDP). The issues with these approaches are mainly two: only shallow networks can be trained with it, and STDP acts mostly as a switch for the connections. These two issues reduce the expressiveness of the network. This training family has been used to solve tasks in supervised learning as [15] in which the authors classified MNIST in a semi-supervised way and in reinforcement learning as [16] in which the authors solve a really simple obstacle avoidance task in which a LEGO robot has to avoid the walls and an obstacle placed in the middle of the environment, [17] in which a lane keeping problem is solved, and [18] in which the control of a robotic arm is performed. In the second training family, an ANN is trained, and then it is converted into SNN. NengoDL [19] is the state-of-the-art framework implementing this training technique. Different tasks were solved with this method, such as [20] in which an obstacle avoidance task is performed, [21] in which the authors teach a network to drive using LiDAR and [22] in which the authors solve the OpenAI Gym atari games [23]. The downside of this approach is that, during the training phase, the expressive potential of the SNNs is not fully exploited, which is supposed to be greater than the ANNs one [1]. The last family uses backpropagation or its approximation to overcome the training issues created by the non-differentiable activation function of the spiking neurons. Some of well known algorithms are e-prop [9], Spikeprop [24], STBP [11] and Slayer [25]. With this family of training algorithms, the SNNs were used in image classification [25,26], in anomalies detection [27] and in reinforcement learning [9,28,29]. A well-known training method called Spatio-Temporal BackPropagation (STBP) [11] overcomes the problem of the non-differentiable spike function using different curves called pseudo-derivative $h(v)$:

$$h(v) = \frac{1}{a} sign\left(|v - v_{thr}| < \frac{a}{2}\right) \tag{7}$$

$$h(v) = \left(\frac{\sqrt{a}}{2} - \frac{a}{4}|v - v_{thr}|\right) sign\left(\frac{2}{\sqrt{a}} - |v - v_{thr}|\right) \tag{8}$$

$$h(v) = \frac{1}{a} \frac{e^{\frac{v_{thr}-v}{a}}}{\left(1 + e^{\frac{v_{thr}-v}{a}}\right)^2} \tag{9}$$

$$h(v) = \frac{1}{\sqrt{2\pi a}} e^{-\frac{(v-v_{thr})^2}{2a}} \tag{10}$$

where a is the peak width of the derivative and, from top to bottom, they represent the derivative of the rectangular function, the polynomial function, the sigmoid function, and the Gaussian function [11]. In our work, we use reset LIF and both the spiking and non-spiking neurons.

1.4. Dynamic Vision Sensor

The Dynamic Vision Sensor (DVS) is a neuromorphic event-based sensor that mimics the eyes of animals. In DVS, each pixel holds a memorized brightness value and emits an event when it detects a change in the log luminance. A DVS does not detect the background; hence it does not send redundant information. Therefore, the output of a DVS is an array of asynchronous and sparse events in which each element is composed of the x–y coordinates of the pixel, its timestamp and polarity, which can be 1 or −1.

In this work, we consider a drone equipped with a DVS camera as an input to the agent, which uses a neural network for learning the environment and take decisions to avoid obstacles. An example of simulated DVS output and its comparison with an RGB image is shown

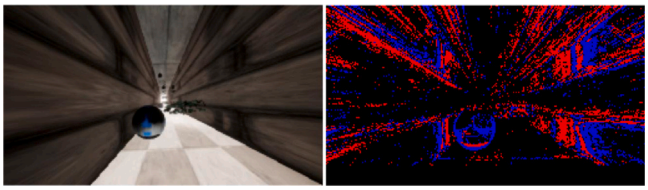


Fig. 2. Comparison between an RGB image and an DVS image.

in Fig. 2. On the right image is shown a DVS image composed of red pixels, the positive events, blue pixels, the negative ones, and black pixels, the background (no events), while the left image is shown the RGB counterpart.

1.5. Neuromorphic platforms

Neuromorphic platforms can be divided into two main categories: Analogue and mixed-signal and digital SNN accelerators. The platforms in the first category are typically more efficient and present a smaller neuron area footprint, given the same neuron model [3,30]. However, the neuron functionality is often technology-dependent and it is implemented by operating individual transistors in their sub-threshold regime; Therefore, it requires a significant engineering effort to be ported to a different technology node. The digital counterpart, instead, implements less complex neuron models; typically a LIF or its derivation [2,4,5,31], and are implemented as equation-solver data paths composed of digital elementary adders and multipliers.

Digital neuromorphic platforms can also be exploited outside the neuromorphic simulation context [32] and allow for fast integration into digital SoCs [13] and technology porting.

Among the different neuromorphic hardware available in the SoA, several common characteristics emerge which induce approximation in the SNN models. Biases are supported in both Loihi [2] and Spinnaker2 [33] while SNE does not have them since their implementation would slow down the simulation and increase the energy consumption of the hardware since the biases force the hardware to update the neurons' internal states each time steps, even if there are not input spikes. Moreover, access to the membrane potential at run time is important since it allows the network to have an increased expressiveness. Access to the internal states is allowed only in Spinnaker2 [33] while in Loihi [2] only at the end of the simulation. In SNE the membrane potential is not accessible. Finally, SNE presents high quantization weights (4 bits) and consumes orders of magnitude less energy. Based on this analysis we can consider SNE to be the worst-case one as it imposes all the mentioned approximations on the neuron model. Hence, the target deployment platform for the quantized spiking neural network presented in this work is an implementation of the Sparse Neural Engine (SNE) presented in [13]. SNE is a fully-digital, non-Von Neumann data-flow architecture with DMA capabilities implementing a programmable number of digital LIF neurons. The accelerator can be integrated into a conventional SoC by connecting it to two dedicated memory ports, and one configuration advanced peripheral bus (APB).

SNE uses an explicit coordinate list (COO) representation encoded on 32 bits to address and consume single events (input feature maps) that can be linearly stored in the main system memory. The architecture in the exam features a dedicated local memory to store up to 256 3×3 4bit-quantized convolutional kernels; the firing threshold and the LIF exponential decay time constant are held by dedicated configuration registers and can be programmed at run time. In this work, we will refer to an SNE configuration with 8 parallel computing engines, each simulating 1024 configurable LIF output neurons.

2. Related work

Since SNNs are described by a dynamic system, they are naturally made for temporal problems such as the ones that RL algorithms solve. In particular, in [17,18,34] the authors use an STDP-based learning rule to solve three OpenAI Gym Atari games [23], an arm-control task, and a lane-keeping problem, respectively. STDP has also been applied to navigation and obstacle avoidance tasks by the authors of [16,35] where an agent has been trained in RL to move in 2D small environments (i.e., a 1000×800 pixels rectangle and an ellipse) avoiding obstacles (in a number of units) using ultrasonic sensors. When it comes to visual-based inputs, like events or RGB cameras, STDP has been widely used to train SNNs for solving classification tasks, like MNIST or DVS-MNIST problems [15,36]. However, to the best of the authors' knowledge, STDP has not yet been applied to solve an obstacle avoidance task leveraging visual-based input sensors, which is the focus of this paper. In future work, we will explore the effectiveness of hybrid STDP and gradient-based approaches for UAV navigation tasks.

To overcome the limitation of applying STDP in more complex RL tasks, other training approaches have been proposed, such as training an ANN model and then converting it into an SNN [37] or using gradient-based training algorithms [11]. Authors of [22,38] trained an ANN in RL and then converted it into an SNN to solve OpenAI Gym games [23]. Similarly, the authors of [21] applied the same methodology to a navigation task using a LiDAR input, and the authors of [20] applied the same methodology – using the NengoDL training framework – to perform an obstacle avoidance task with a drone using an event-camera as an input. To the best of our knowledge, [20] is the only work in the SoA solving a DVS-based obstacle avoidance task with SNNs and RL. In [20], the authors validate their approach by flying a UAV in a photo-realistic simulated environment (using AirSim [39] flight simulator) composed of several lanes with static obstacles. The simulated agent is a UAV constantly forward along the lane. The ANN (and yet the NengoDL converted SNN) output correspond to the five possible actions: go left, go right, go down, go up, and maintain the course. The input of the network is three consecutive DVS frames. The ANN network is trained using D2QN RL approach, which then is converted to an SNN using NengoDL. The latter restricts the authors in mapping the three DVS frames in different input channels — de facto neglecting the temporal capabilities of SNNs. In the proposed paper, we restrict ourselves to the ANN and SNN network topology as the one presented in [20], and we maintain the same RL problem (RL method, action space, and observation space) in a similar simulation scenario. To overcome the limitation of [20] and demonstrate the challenges in applying SNNs into real digital neuromorphic hardware, in our work we introduced an RL method based on STBP which allows preserving the temporal capabilities of SNNs during the training, we modified the SNN model and introduced the quantization to match the capabilities of neuromorphic hardware accelerators compatible with the application domain. In [20], the authors validate their approach by flying a UAV in a photo-realistic simulated environment (using AirSim [39] flight simulator) composed of several lanes with static obstacles. The simulated agent is a UAV constantly forward along the lane. The ANN (and yet the NengoDL converted SNN) output correspond to the five possible actions: go left, go right, go down, go up, and maintain the course. The input of the network is three consecutive DVS frames. The ANN network is trained using D2QN RL approach, which then is converted to an SNN using NengoDL. The latter restricts the authors in mapping the three DVS frames in different input channels — de facto neglecting the temporal capabilities of SNNs. In the proposed paper, compared with the above-mentioned paper: we restrict ourselves to the ANN and SNN network topology as the one presented in [20], we maintain the same RL problem (RL method, action space, and observation space) in a similar simulation scenario. We, however, made some improvements: (i) we chose a more realistic DVS-camera simulated model [40] and (ii) we made the simulated environment more complex. We evaluated the

Table 1
Networks architecture.

	CNN	SCNN	eSCNN	qeSCNN	pqeSCNN
Input channels	3	2	2	2	2
CONV1	(16, 8, 4)	(16, 8, 4)	(16, 8, 4)	(16, 8, 4)	(16, 8, 4)
CONV2	(32, 8, 4)	(32, 8, 4)	(32, 8, 4)	(32, 8, 4)	(32, 8, 4)
FC1	512	512	512	512	512
FC2	5	5	5	5	5
Bias	✓	✓	✗	✗	✗
Type of hidden neurons	ReLU	Hard reset - LIF	Hard reset - LIF	Hard reset - LIF	Hard reset - LIF
Type of output neurons	Linear	Non-spiking LIF	Non-spiking LIF	Non-spiking LIF	Non-spiking LIF
Quantization	✗	✗	✗	Full quantized	No last layer

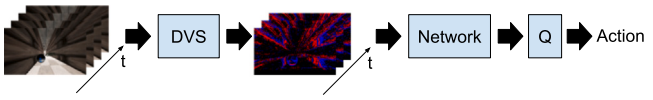


Fig. 3. Pipeline of our setup. The RGB frames are converted into DVS frames which are fed into the network. The output of the network is the Q-function which is used to choose the action that the drone has to perform.

complexity of the simulated environments by means of the performance achieved by a random agent. In [20], a random agent reaches the end about 25% of the time, while in our simulated environment, the random agent reaches the end of the lane 0% of the time. When an ANN is trained as a proxy to train the SNN, the performance attainable by the SNN is constrained by the performance achieved by the ANN [19]. Indeed the SNN network has to have the same network topology as the ANN one, and in the best-case scenario, it can perform as well as the ANN one (since it is a conversion of it). To overcome the expressiveness limitation of the network during the training, gradient-based training algorithms for SNNs have been proposed.

In these algorithms, the spike function's non-derivative problem is bypassed using a pseudo-derivative. In [29], the authors compare the performance of an ANN, an SNN, and an SNN converted from an ANN in 17 OpenAI Gym games. The results show that the SNN outperforms the ANN on 12 games. The ANN outperforms the SNN in four games. In one game (Tennis), all three networks perform the same. Interestingly, the SNN converted from an ANN never outperforms the others — as early discussed, the converted SNN from the ANN, in the best case, can only perform at most as the ANN, and discards temporal SNNs capabilities. In [28], the authors use a gradient-based training algorithm implemented with the SpyTorch tool [41] to solve two OpenAI Gym games [23], namely the CartPole and Acrobot. The authors validated their network by executing it in the Intel Loihi board [2]. To do so, the author proposed a quantization-aware training algorithm to match the Loihi weight resolution and used the membrane potential on the last layer to select the action.

In [9], the authors proposed a new training method in which the gradient is accumulated during the inference in the training phase and then applied to the network at the end of the inference. This method was tested in two OpenAI Gym games [23], and works only with recurrent SNNs. As a matter of fact, gradient-based approaches have been applied only in conjunction with RL to OpenAI gym games and never with DVS input in photorealistic environments. To stress the gradient-based SNNs training algorithms to more realistic scenarios, in this manuscript, we applied it to solve the problem of a UAV navigation and obstacle avoidance task with DVS-camera input as the one presented in [20], taking the same ANNs topology as SoA reference, and we evaluated the approximations which need to be added to fit the network into an SoA low-power embedded neuromorphic hardware called SNE [13] designed for nano-drones and cyber–physical system applications [6].

3. Methodology

In this work, we want to evaluate an SNN inside an RL algorithm for an obstacle avoidance task, comparing it with an ANN-based version [20]. Fig. 3 shows the pipeline used for the comparison. The RGB frames are fed into the DVS simulator, which converts them into DVS images. These images are the input of the neural networks which model the Q-function and give, as output, a Q-array. The action taken by the agent is the one that corresponds to the maximum value in the Q-array.

After this comparison, we create a framework that simulates the behaviour of SNE in which the membrane potentials and the weights are quantized. In the end, we design a new SNN in a way that can fit inside SNE [13].

Our methodology consists of a set of training algorithms and SNN topologies progressively meeting the constraints imposed by the hardware target implementation and by the RL task.

3.1. Spiking neural network and DVS input model

In this work, we focus on SNN for reinforcement learning tasks. This implies some relevant considerations about the network and input model for SNN used for classification.

Indeed, SNNs used in classification tasks use multiple ticks in which the data fed to the networks are the same and measure the spike activity of the output to assign a class. This strategy lets the networks evolve and reach a steady state. Recent works show how this latency can be tuned as a hyperparameter [42]. Hence, the input of the network has to be shown for a certain amount of time called ticks (N) to allow the network to reach a steady state. Normally the longer the persistence of the input, the higher the accuracy of the network. To keep the same input for a certain amount of time, two strategies can be used: (i) In the case of an event-based input such as a sequence of DVS frames, the sequence to be classified is recorded and repeated for a predefined number of times [43]; (ii) In case the input is an RGB/grayscale frame, it is kept fixed for a certain amount of ticks allowing the encoding strategy to convert the frame into spike trains. Examples of the second method are (i) in [44] the input images are translated in spike trains where the pixels values are proportional to the probability of firing and (ii) in [9] the pixels value are used as feeding current of the neurons in the input layer.

In this work, we focus on reinforcement learning using an event-based input. This input is time-varying and encoded as spikes because of the use of a DVS camera simulator.

However, in a reinforcement learning case where the network has to observe and react to the environment to estimate a Q-function, there is no defined sequence to be classified. Hence, recording and repeating a scene is not feasible as the network processes the input events as they are produced.

Also, the network has a constrained time to converge because the agent must decide to navigate the environment. For this reason, we use the spikes generated by the DVS directly as input of the network to reduce latency. A sequence of DVS (differential) images is provided as staked frames to the network. The ticks of the network (N) are the same number of the DVS frames fed into the network. The SNN

will process all the stacked frames to produce an output estimation of the Q-function. The number of stacked frames is a design parameter and should be kept small to avoid increasing the latency. As reported in the experimental section, we found that we can achieve acceptable accuracy with a minimum number of 3 stacked frames.

Since SNNs can have only positive spikes and they are described by a dynamic system in which the time is explicit, we can make the following considerations: (i) The negative DVS events are converted into positive ones. They are fed into the network in a different channel. Hence 2 input channels for frames are needed; (ii) The stacked frames used for detecting the status of the environment are fed as temporal sequence samples; hence the number of time steps in the SNNs is equal to the number of stacked frames: $N = S$. The stacked frames used for detecting the status of the environment are fed as temporal sequence samples; hence the number of time steps (*ticks*) in the SNNs is equal to the number of stacked frames: $N = S$ which is equal to 3. In this process, every single frame is shown to the network for 1 tick.

The network used for tackling the RL task comprises 2 types of neurons: (i) The spiking LIF with hard reset in the hidden layer and (ii) The non-spiking LIF in the output layer. Both neuron types have a bias. We will consider this network as the SNN baseline for our study and refer to it in the experimental results as SCNN.

The SNN model is then modified to match the capabilities of real-life neuromorphic hardware to study the feasibility of equipping a real agent with this type of network. Also, quantization must be applied to reduce the memory footprint and match the target computing architecture of an embedded device [12].

Neural network topologies can feature billions of parameters, typically represented as single-precision floating-point values [45]. This choice is dictated by the need to ensure the numerical stability of the algorithms used to train the network. The memory footprint of such a neural network featuring floating point parameter representation is typically in the order of several tens of GB. It might not fit the amount of memory hosted by embedded computing platforms where such a network would be deployed.

Straightforward quantization approaches can reduce the precision moving from 32 bits or 64 bits floating point representation to a smaller 16 bits float representation. More aggressive quantization strategies can start from the full precision representation of the network parameters and target low bitwidth parameters representation like 8, or even sub-byte integer representation [46]. In the context of this work, we will target 4-bit quantized network weight parameters and 8-bit quantized internal membrane potential representation to match the targeted SNE neuromorphic accelerator. The intermediate feature map, i.e., the output produced by a neural network layer, is inherently quantized to 1-bit activation in SNNs. Indeed, contrarily to CNN continuous-valued activation, the activation function of SNN layers produces binary events indicating that the neuron membrane potential has exceeded a threshold at the corresponding time instant. More details about the quantization strategy of both weights and neuron dynamics are reported in Section 3.4.

3.2. The adopted reinforcement learning model

In this work, we consider a well-known easy to use RL family known as deep Q-learning, which is a set of model-free, value-based and off-policy algorithms in which the goal is to maximize the expected reward [7] that is the reward that the agent expects to collect the next steps. In particular, with reference to Fig. 1, the goal of Q-learning is to maximize the action-value function $Q(s,a)$. This function, given a state s, assigns a score to all the possible actions a that the agent can perform:

$$Q^{\pi}(s,a) = \mathbb{E}_{s_0=s,a_0=a,\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r_t\right] \tag{11}$$

where the action-value function $Q(s,a)$ has to be referred to as a policy π, and it is computed as the expectation of the future rewards r sampled from a trajectory τ and discounted by a factor γ.

In particular, in the DQN algorithm, the Eq. (11) can be rewritten as:

$$Q^{\pi}(s,a) \approx r + \gamma \max_{a'} Q^{\pi}(s',a') = Q^{\pi}_{tar}(s,a) \tag{12}$$

where $Q^{\pi}_{tar}(s,a)$ is the target action-value function and $Q^{\pi}(s',a')$ is produced by the neural network that is used for choosing the actions.

This algorithm has two main limitations: (i) The network that estimates the target is the same as the one that estimates the actions, causing the instability of the target function [47] and (ii) The network that estimates the next action is the same as the one which estimates the $Q^{\pi}_{tar}(s,a)$, leading to an optimistic prediction [48].

To face the first problem, we adopted an approach proposed in [47] where a lagged copy of the network is used. This implies using two networks: One for the evaluation task that is described by its own set of parameters (φ) and the network used for choosing the action (θ) [47]. To face the second problem, we evaluate the next action (a') with the θ network [48], which is the Double DQN (D2QN) approach. Hence, Eq. (12) becomes:

$$Q^{\pi}_{tar}(s,a) = r + \gamma Q^{\pi_{\varphi}}(s', \max_{a'} Q^{\pi_{\theta}}(s',a')) \tag{13}$$

The loss function will be computed as mean square error between $Q^{\pi}_{tar}(s,a)$ and $Q^{\pi_{\theta}}(s,a)$.

In general RL algorithm scalability is limited by the training time which depends on the time it takes to simulate a step in the simulated scenario. To overcome this limitation, two main approaches have been proposed: The acceleration of the simulation time [49,50]; and the usage of multiple environments running in parallel [7,50]. The D2QN algorithm is an off-policy algorithm and thus it can leverage only the acceleration of the simulation time to speed up the training time. In addition, the D2QN algorithm learns a discrete action space, which can limit its scope of applicability.

3.3. The adaptation of STBP for Q-function estimation

In the RL task we deal with in this work, the role of the SNN is to estimate a Q-function and use the estimation to take actions. This function in our problem of navigation in a photorealistic environment is a complex one that a network should describe with high expressiveness.

For training the SNNs, we used a well know training algorithm called STBP [11], which uses the pseudo-derivative described by Eq. (7) and is designed for training SNNs in classification tasks and is based on the average output activity. In Table 2 are reported 3 examples for each output strategy shown in the following. The examples are computed with a number of ticks (N) equal to 3. The first row is the plot of the output activity, while the second row is the output activity "translated" in spikes. The following 4 rows are the 4 strategies that can be used as the output of a spiking neural network. The first strategy is the mean output activity which presents a low expressiveness since the output value will be in the rage $[0,1]$ and because the possible output levels n_{ol} are strictly dependent on the number of ticks N of the network: $n_{ol} = N + 1$ (Table 2 - Output i). A possibility to increase the expressiveness of the network is to consider the "time to spike" of the last layer. In this case, the output will be described by: $n_{ol} = 2^N$ (Table 2 - Output ii). As discussed in Section 3.1, N is equal to the number of frames fed into the network. As such, we want to keep it as small as possible since a high number of frames would lead to increased latency and increase the amount of computation. A possible way to overcome the limited output levels is to use the membrane potential $v(t)$ of the spiking neurons in the output layer. In this case, the output can reach the values in the interval $[-\infty, v_{thr}]$, and the number of output levels will increase up to: $n_{ol} = \infty$ (Table 2 - Output iii). With this implementation, however, the Q-function cannot reach a value greater

Table 2
Three examples of neuron output values computed with different strategies. In these examples, we use a number of ticks (N) equal to 3.

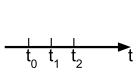
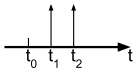
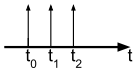
		Case 1	Case 2	Case 3
Output	Activity			
	Spikes	$z_0 = 0;\ z_1 = 0;\ z_2 = 0$	$z_0 = 0;\ z_1 = 1;\ z_2 = 1$	$z_0 = 1;\ z_1 = 1;\ z_2 = 1$
	i: $1/N * \sum_i z_i$	0	0.67	1
	ii: $\sum_i 2^i * z_i$	0	6	7
	iii: Eqs. (2) & (5)	$v(t) < v_{th}$	v_{th}	v_{th}
	iv: Eq. (2)	$-\infty < v(t) < \infty$	$-\infty < v(t) < \infty$	$-\infty < v(t) < \infty$

Table 3
Comparison between the features of a full customize neuron and the neuron in SNE.

Features	LIF Neuron	SNE LIF Neuron
Bias	✓	✗
No-spiking neuron	✓	✗
Type of reset	Hard & Soft	Hard
Access to $v(t)$	✓	✓
Acces to $z(t)$	✓	✓

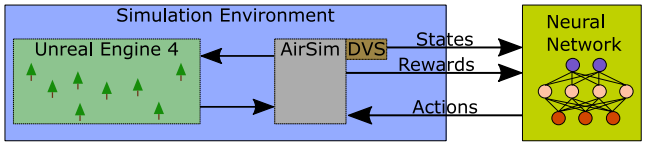


Fig. 4. The full photorealistic pipeline. The blue box is the simulation environment that is composed of Unreal Engine 4, AirSim and the DVS while the yellow box is the network.

than v_{thr}. To tackle also this problem, we use a non-spiking LIF as output. Hence, the output membrane potential can reach the values in interval $[-\infty, \infty]$ and the output levels will be $n_{ol} = \infty$ (Table 2 - Output iv).

3.4. Spiking Neural Network on SNE embedded neuromorphic accelerator

The SNN model described in the previous section has been modified to match the requirements of the SNE accelerator. The first adaptation concerns the neuron model, and the second relates to the quantization of weights and membrane potential. The differences between the software neuron model and the implementation available in SNE are reported in Table 3. To match the SNE neuron model, we first modified Eq. (3) to remove the bias:

$$I_i^n(t) = \sum_{j=1}^{l(n-1)} w_{ij}^n z_j^{n-1}(t-1) \tag{14}$$

In the following, we will refer to this network as eSCNN.

To deploy SCNNs onto the embedded neuromorphic hardware, the network parameters (i.e. weights and intermediate feature maps) must be represented with a bit-width supported by the target architecture.

On SNE, the weights are represented as 4-bit signed integer values. Moreover, compared to traditional ANNs, SCNNs present an internal state variable. In the case of SNE, the LIF membrane potential, i.e. the internal neuron state variable, is represented on 8 bits. During deployment, the neural network weights and the membrane potential must be modelled with an adequate level of numerical precision, i.e. they must be quantized to 8 bits.

To perform this task, we use Quantlib [12], which has been extended to support the activation function implemented on the target neuromorphic HW, i.e. the LIF function implemented on SNE. Specifically, Quantilib supports the quantization of the weights to a desired numerical precision. In the following, we will refer to the full quantized, SNE-deployable network as qeSCNN. This network is composed of 2 different neurons: the non-spiking neurons used for the last layer and the spiking neurons for all the other layers. In this configuration, all the weights and the membrane potential are quantized according to the hardware constraints.

In our experiments, we also implemented a version of the SNN where the last layer is implemented in the general-purpose microcontroller of SNE. Therefore there is no need to restrict its numerical precision to 8 bits, i.e. ($n_{ol} = 256$). We called this network pqeSCNN. Such a choice has been made to account for a possible loss of expressiveness of the SNN due to the low number of time steps used in the decision layer. A summary of the networks compared in this work is shown in the 1.

To evaluate the inference accuracy achievable on the target hardware platform, we modelled the exact behaviour of the LIF neurons implemented on SNE. Specifically, we modelled the 8-bit membrane potential decay performed on the real hardware using look up table (LUT), as well as the neuron membrane potential update in the occurrence of an incoming spike and the membrane potential reset mechanism once the threshold is exceeded, which restores the membrane potential to its rest value. This goal has been achieved by extending the base neural network classes defined by the Quantlib framework [12]. The synaptic connectivity supported by SNE is of convolutional and fully connected type.

4. Training and evaluation framework

In this section, we describe the simulator we used for training the drone to perform the obstacle avoidance task, outlining the environment characteristics, the RL parameters and the configuration of the DVS.

4.1. Simulation environment and DVS model

A simulator is used to model agent actions in the environment and build a dataset for the RL obstacle avoidance task. In this work we use AirSim [39], built on top of Unreal Engine 4 graphic engine [51]. AirSim uses Unreal Engine for the physics simulation and the rendering. Being photo-realistic makes it possible to use the trained network in the real world with an acceptable performance drop [52].

In the past years, a lot of effort was made to create a DVS simulator [40,53–55], and in this work, we used v2e [40] since it is realistic as the tool models the DVS dynamic and noise.

We integrated DVS in AirSim so that the images generated by AirSim are processed by v2e and then passed to the RL agent as input.

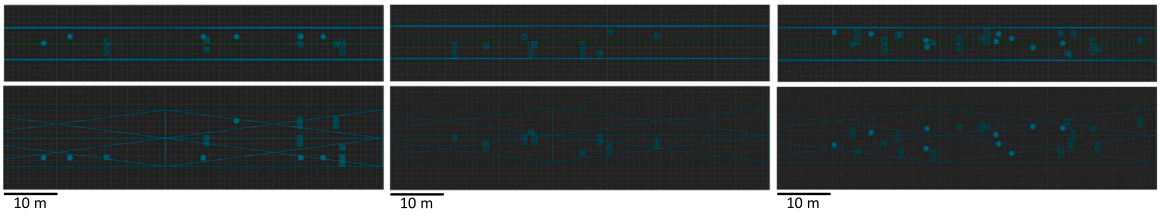
4.2. Training and evaluation setup

This section describes the training and evaluation pipeline presented in Fig. 4. The blue box is the environment simulator, while the yellow one is the network trained with the D2QN algorithm. The network only has access to the observations of the environment and the rewards that measure how well the network performs.

In this work, we create a setup similar to [20], creating 3 different 70 m-long lanes in Unreal Engine 4 along the y-axis in which we place several obstacles of different sizes and shapes.

Table 4
Simulation parameters value and description.

	Parameters	Values	Description
DVS	Positive Threshold	0.2	Nominal threshold of triggering positive event in log intensity
	Negative Threshold	0.2	Nominal threshold of triggering negative event in log intensity
	Sigma Threshold	0.03	Standard deviation of threshold in log intensity
	Cutoff Frequency [Hz]	300	3 dB cutoff frequency of DVS photoreceptor
	Leak Rate Frequency [Hz]	0.01	Leak event rate per pixel
	Shot Noise Rate Frequency [Hz]	0.001	Shot noise rate
Reward	γ_y	1	Weight for the distance reward
	γ_a	−0.1	Weight for the action reward
	γ_c	−10	Weight for the collision reward
	M	250	Maximum amount of actions that the drone can take
Simulation	γ	0.99	Discount factor
	Staked Frames	3	Number of staked frames feed to the neural network
	# Train Games	5 000	Number of training games
	# Test Games	100	Number of testing games per lane



(a) Training lane. (b) Lane 1. (c) Lane 2.

Fig. 5. The position of the obstacles in the 3 lanes. The first row is the representation from the top, while the second is the representation from the left.

The lanes created are shown in Fig. 5. From left to right: (a) Training lane, used in the training and in the evaluation with 16 obstacles; (b) Lane 1, used for the evaluation, featuring 9 obstacles, and (c) Lane 2, used for the evaluation, featuring 25 obstacles. The data exchanged between the environment and the agent are the observations and the rewards. Since the observations are composed of 3 pre-processed DVS frames, we used a tool called v2e [40] that simulates the real DVS behaviour. All the DVS parameters and the simulation settings can be found in the Table 4. This tool takes, as input, the images gathered from AirSim, and it returns a list of tuples in which the first entry is the timestamp, the second and the third are the x and y coordinates of the pixel, and the last one is the polarity. The polarity can be 1 or −1, depending on the variation of the log-luminance. The second data exchanged by the environment are the rewards. The rewards are computed as follows:

$$R(s,a) = \gamma_y \Delta y - \gamma_c C - \gamma_a A \tag{15}$$

where R is the reward of the state s taken action a, γs are some regularization factors that are used to weight the sum, Δy is the distance between the goal and the drone, C is a flag that indicates if the drone has collided and A is the action taken. A is computed as:

$$A = \begin{cases} 0 & \text{if index action is 4} \\ 1 & \text{otherwise} \end{cases} \tag{16}$$

$$C = \begin{cases} 1 & \text{if the drone has collided or } \#A > \text{M} \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

where $\#A$ is the number of actions done and M is the maximum number of actions the drone can take. The game ends when the drone reaches the goal or when $C = 1$.

The agent communicates the decision taken through the actions, and this environment is characterized by a discrete actions space in which the actions are the following:

- 0: avoid left.
- 1: avoid right.
- 2: avoid down.
- 3: avoid up.
- 4: maintain the course.

The described framework has been used to obtain the results presented in the next section.

5. Results

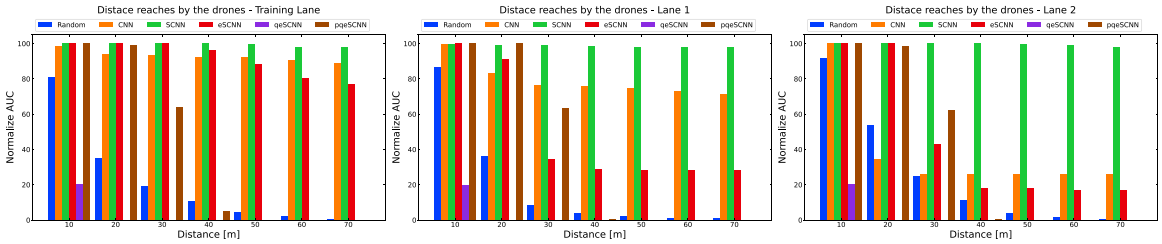
5.1. Neural networks performance

In this section, we compare the performance of the trained networks. To evaluate the networks, we compute the *Normalized Area Under the Curve* (Normalize AUC) and the total amount of drones that reach the goals. The AUC is computed as:

$$AUC = \sum_{rd} d \tag{18}$$

where rd is the distance, a certain number of drones d reached. The Normalize AUC is the AUC divided by the width of the bins.

Fig. 6 reports the performance of the networks in each 10 m bin for all the three lanes. We consider a random agent (blue bars) that we use as baseline. In Table 5, we report the Normalized AUC values and the percentage of drones reaching the final goal of 70 m. The SCNN (green bars) is the best network since it reaches the end of all the 3 lanes 98% of the time while the CNN (orange bars) 89% of the time in the Training lane, 70% in Lane 1, and 17% in Lane 2. This is due to the better capability of the SCNN to solve temporal tasks and shows that the SNNs generalize better than the ANNs counterpart. The eSCNN (red bars) has the same architecture as the SCNN, except for the presence of biases, and it reaches the goals of just 77%, 28%, and 17% of the cases in Training lane, Lane 1, and Lane 2 respectively. We conclude that the biases are relevant for this task because they increase the expressiveness of the network. In general, eSCNN reaches the goals a lower number of times than the CNN, but in the Training lane and in Lane 2 they have a similar Normalized AUC. In these two lanes, the two networks show similar behaviour. In Lane 1, the eSCNN has a big drop in the third bin, showing high difficulties to pass through that sequence of obstacles. This is well-shown in Fig. 6(b) and in Table 5 and is due to the lack of biases which decreases the expressiveness of the network. Both the



(a) Training lane. (b) Lane 1. (c) Lane 2.

Fig. 6. Performance of the networks in 3 lanes. It is shown the Normalize AUC computed with bins of 10 m.

Table 5
Networks performance.

	Normalized AUC			Drones reaching the goal		
	Training lane	Lane 1	Lane 2	Training lane	Lane 1	Lane 2
Random	21.70	19.83	26.63	0%	1%	0%
CNN	92.79	78.94	37.74	89%	70%	26%
SCNN	99.34	98.51	99.5	98%	98%	98%
eSCNN	91.67	48.30	44.64	77%	28%	17%
qeSCNN	2.86	2.83	2.86	0%	0%	0%
pqeSCNN	38.20	37.61	37.23	0%	0%	0%

qeSCNN (purple bars) and the pqeSCNN (brown bars) networks are post-training quantized. The difference between the two networks lies in the quantization of the last layer. Indeed, in the qeSCNN, the last layer is quantized. As a result, the network shows the worst overall score with respect to the other networks: it does not even reach the second bin (Fig. 6) while the pqeSCNN, in which the last layer is not quantized, reaches the fourth bin in the Training lane and the third in both Lane 1 and Lane 2. The poor performance of the qeSCNN is due to the low expressiveness of the last layer, as it has 256 levels to represent the Q-function. On the other side, the pqeSCNN version, where the output layer is executed on a RISC-V-Based 8-core cluster, can exploit the full dynamic range. The first two bins show accuracies comparable with SCNN and eSCNN (Fig. 6). However, a performance degradation from the third bin happens due to the approximations introduced by the post-training quantization. In the end, we can see that even if the random agent reaches the sixth bin while the pqeSCNN the fourth one, the latter has a higher Normalized AUC (Table 5). This happens because the random agent shows remarkably low performance from the beginning while the pqeSCNN from the third bin (Fig. 6).

5.2. Spiking Neural Network Footprint

In this section, we present energy and latency estimates of the quantized SNN when running on the digital embedded neuromorphic platforms presented in [13]. Estimates have been calculated based on the network input and hidden layers' spike activity. The input activity estimates are measured on the entire dataset used to train the network on the proposed RL task. Fig. 7 reports three relevant metrics derived from the quantized SNN spike activity. Fig. 7(a) shows that the activity increases in the deeper layers of the network. This result is expected as the network has not been penalized for spiking high activity during the training. Therefore, the network has not been forced to prune redundant synaptic connections, which might increase in the deeper layers [56]. As the accelerator in exams is designed to achieve energy-to-activity proportionality, the network activity determines the inference energy consumption and inference latency proportionally.

Fig. 7(b) reports the energy consumed by the SNE platform on each network layer. Convolutional layers implement a smaller number of synaptic operations thanks to the sparsity of the first layers. Therefore, the energy to execute such layers is, on average, lower than the one required to execute linear layers.

The total average energy required to perform a complete inference is 0.62 uJ/inf. Note that the energy consumed during an inference strongly depends on the activity of the network. On this data set, the minimum estimated energy consumed on SNE is 0.41 uJ/inf, while the maximum is 0.85 uJ/inf. Depending on the input stream of events captured in the event-camera field of view, there is a variation. During the RL training phase, the network has not been trained to reduce the spike activity of hidden layers. See Fig. 7(a).

Table 6
Number of the total operation and inference energy for the proposed approach and the baseline CNN executed on a reference state-of-the-art hardware accelerator [59]. Estimates are based on the highest energy per inference reported by the author.

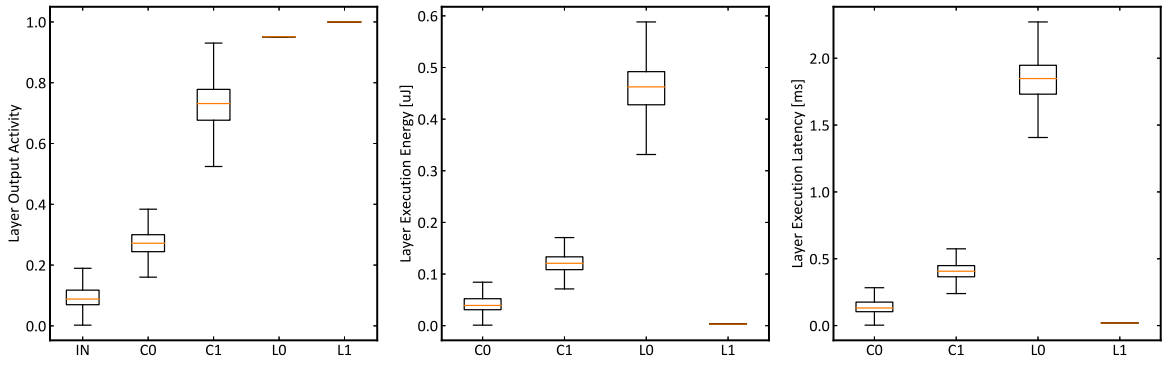
Network	Number of operations	Inference energy
CNN	23.2 MOP	24.11 μJ[a] (3.6 μJ[b])
qeSCNN	2.9 MOP	0.62 μJ
pqeSCNN	2.9 MOP	0.66 μJ

[a] Energy per inference estimate based on the number of MAC executed on the reference hardware accelerator [59].
[b] Energy per inference estimate on the reference hardware accelerator with energy cost scaled to the SNE technology.

Similar estimates can be computed for the inference latency. Fig. 7(c) reports estimates of the inference latency when the proposed SNN is executing on SNE. The average inference latency is 2.4 ms, the minimum inference latency is 1.7 ms, and the maximum inference latency is 3.14 ms.

Such results demonstrate that it is possible to solve the navigation and obstacle avoidance task at a frame rate that is comparable to or higher than what was reported for SoA embedded deployment of resource-constrained neural networks in small drones such as Dronet [57]. Dronet has been deployed on an embedded SoC featuring a RISC-V cluster [58] and optimized to reach 15 mJ/inf. Compared to such approach, the proposed RL strategy coupled with an efficient deployment on dedicated neuromorphic hardware can lead to more than three orders of magnitude energy efficiency improvement.

To perform an energy comparison with the CNN, we considered state-of-the-art CNN implementation on a dedicated programmable hardware accelerator capable of executing both compact and sparse DNNs [59]. To have a fair comparison, both the CNN and SNN workloads have been decomposed into elementary operations, i.e., multiplication or addition operations. Results are reported in Table 6. In the case of CNN, operations are MACs (multiply and accumulate) performed to compute the value of each output pixel. In the case of



(a) SNE Layer activity. (b) inference energy. (c) Inference latency.

Fig. 7. The Convolutional layers are indicated as "Cn", Linear layers are indicated as "Ln", and "IN" indicates the network input activity from the event camera.

SNN, additions originate from the sparse weight accumulation process on the membrane potential, while multiplications are performed to implement the membrane potential exponential decay. When executing an inference on the proposed network topology, the SNN variant executes 8× fewer operations than the CNN variant, thanks to its sparse nature.

Results in Table 6 highlight that when SNN inference is executed on dedicated hardware capable of efficiently performing sparse operations, the resulting inference energy cost is lower than the CNN. The energy cost remains a factor of 6× lower for SNE even when considering the energy consumed by the reference CNN accelerator in [59] with energy numbers scaled to SNE technology.

6. Conclusions

In this work, we addressed the issue of training the SNNs in RL for solving a real obstacle avoidance task. The network is designed to fit in state-of-the-art neuromorphic hardware called SNE.

- We created a full photorealistic pipeline for training the networks in an obstacle avoidance task using Unreal Engine 4 as environment simulator, AirSim as UAV simulator and v2e for converting the RGB frame collected from UE4 to DVS frames.
- We trained an SNN, without converting it from the ANN, to solve the obstacle avoidance task. To do so, we modify the Spatio-Temporal BackPropagation SNN training method, allowing the use of the membrane potential of the output neurons for evaluating the action that has to be performed in the next step.
- We create a plugin for QuantLab that allows us to quantize the weights and the membrane potential of the SNNs to simulate the behaviour of the network in real neuromorphic hardware.
- We compared the CNN trained in RL with the SCNN, eSCNN, qeSCNN and pqeSCNN. From this comparison, we showed that the best network is the SCNN since it reaches the end 98% of the time in all lanes, while the CNN reaches at most 89% in the lane in which it was trained. Furthermore, we train a new network that fits the hardware constraint (eSCNN), which has no biases. Results show that the biases have a considerable impact on our network performance. Indeed, without them, we experience a drop in performance in all the 3 lanes. Finally, another relevant aspect is the quantization of the last layer. Indeed, the post-train quantization of the last layer causes a non-acceptable reduction of the expressiveness of the network — thus a solution where the quantization of the last layer is performed at full precision or in software in the host processor is more effective.
- We evaluated the energy consumption and the latency of the network as if it was developed in SNE. The energy consumption and the latency are strongly linked to the spike activity, and we did not train the network to minimize it. This analysis shows that the average energy required to perform a complete inference is 9.73 uJ/inf, while the latency is 33.35 ms.
- We compared the energy consumption of both the 2 embedded networks and the CNN, showing that even if the SNNs were not trained for having a sparse activity, both the qeSCNN and the pqeSCNN use 6× less energy compared to the CNN.

CRediT authorship contribution statement

Luca Zanatta: Conceptualization, Methodology, Software, Writing – original draft, Writing – review & editing. **Alfio Di Mauro:** Software, Writing – original draft. **Francesco Barchi:** Supervision. **Andrea Bartolini:** Supervision, Writing – review & editing. **Luca Benini:** Supervision. **Andrea Acquaviva:** Supervision, Writing – review & editing, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

This work was supported in part by the Italian Ministry for Education, University and Research (MIUR) under the program "Dipartimenti di Eccellenza" (2023–2027), the HE EU DECICE project (g.a. 101092582), and the Edge-AI (g.a. 101097300).

References

[1] W. Maass, Networks of spiking neurons: the third generation of neural network models, Neural Netw. 10 (9) (1997) 1659–1671.
[2] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S.H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, et al., Loihi: A neuromorphic manycore processor with on-chip learning, IEEE Micro 38 (1) (2018) 82–99.
[3] S. Moradi, N. Qiao, F. Stefanini, G. Indiveri, A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs), IEEE Trans. Biomed. Circ. Syst. 12 (1) (2017) 106–122.
[4] M.V. DeBole, B. Taba, A. Amir, F. Akopyan, A. Andreopoulos, W.P. Risk, J. Kusnitz, C.O. Otero, T.K. Nayak, R. Appuswamy, P.J. Carlson, A.S. Cassidy, P. Datta, S.K. Esser, G.J. Garreau, K.L. Holland, S. Lekuch, M. Mastro, J. McKinstry, C. di Nolfo, B. Paulovicks, J. Sawada, K. Schleupen, B.G. Shaw, J.L. Klamo, M.D. Flickner, J.V. Arthur, D.S. Modha, TrueNorth: Accelerating from zero to 64 million neurons in 10 years, Computer 52 (05) (2019) 20–29, http://dx.doi.org/10.1109/MC.2019.2903009.
[5] B.V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A.R. Chandrasekaran, J. Bussat, R. Alvarez-Icaza, J.V. Arthur, P.A. Merolla, K. Boahen, Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations, Proc. IEEE 102 (5) (2014) 699–716.

[6] A. Di Mauro, M. Scherer, D. Rossi, L. Benini, Kraken: A direct event/frame-based multi-sensor fusion SoC for ultra-efficient visual processing in nano-UAVs, 2022, arXiv preprint arXiv:2209.01065.

[7] W.L. Keng, L. Graesser, SLM lab, 2017, GitHub repository, GitHub, https://github.com/kengz/SLM-Lab.

[8] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, M. Hutter, Learning agile and dynamic motor skills for legged robots, Science Robotics 4 (26) (2019) eaau5872.

[9] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, W. Maass, A solution to the learning dilemma for recurrent networks of spiking neurons, Nat. Commun. 11 (1) (2020) 1–15.

[10] A. Bachrach, Skydio autonomy engine: Enabling the next generation of autonomous flight, in: 2021 IEEE Hot Chips 33 Symposium, HCS, 2021, http://dx.doi.org/10.1109/HCS52781.2021.9567400.

[11] Y. Wu, L. Deng, G. Li, J. Zhu, L. Shi, Spatio-temporal backpropagation for training high-performance spiking neural networks, Front. Neurosci. 12 (2018) 331.

[12] M. Spallanzani, G.P. Leonardi, L. Benini, Training quantised neural networks with STE variants: the additive noise annealing algorithm, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 470–479.

[13] A. Di Mauro, A. Suravi Prasad, Z. Huang, M. Spallanzani, F. Conti, L. Benini, SNE: an energy-proportional digital accelerator for sparse event-based convolutions, in: Design, Automation and Test in Europe Conference, DATE 2022, EC, 2022, http://dx.doi.org/10.3929/ethz-b-000543342.

[14] M. Lapan, Deep Reinforcement Learning Hands-on, Packt publishing, 2020.

[15] P.U. Diehl, M. Cook, Unsupervised learning of digit recognition using spike-timing-dependent plasticity, Front. Comput. Neurosci. 9 (2015) 99.

[16] S.A. Lobov, A.N. Mikhaylov, M. Shamshin, V.A. Makarov, V.B. Kazantsev, Spatial properties of STDP in a self-learning spiking neural network enable controlling a mobile robot, Front. Neurosci. 14 (2020) 88.

[17] Z. Bing, C. Meschede, K. Huang, G. Chen, F. Rohrbein, M. Akl, A. Knoll, End to end learning of spiking neural network based on r-stdp for a lane keeping vehicle, in: 2018 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2018, pp. 4725–4732.

[18] J.C.V. Tieck, P. Becker, I. Peric, J. Kaiser, M. Akl, D. Reichard, A. Roennau, R. Dillmann, Learning target reaching motions with a robotic arm using dopamine modulated STDP, in: 18th IEEE International Conference on Cognitive Informatics and Computing, 2019.

[19] D. Rasmussen, NengoDL: Combining deep learning and neuromorphic modelling methods, 2018, pp. 1–22, arXiv 1805.11144, URL http://arxiv.org/abs/1805.11144.

[20] N. Salvatore, S. Mian, C. Abidi, A.D. George, A neuro-inspired approach to intelligent collision avoidance and navigation, in: 2020 AIAA/IEEE 39th Digital Avionics Systems Conference, DASC, IEEE, 2020, pp. 1–9.

[21] A. Shalumov, R. Halaly, E.E. Tsur, Lidar-driven spiking neural network for collision avoidance in autonomous driving, Bioinspiration Biomim. 16 (6) (2021) 066016.

[22] D. Patel, H. Hazan, D.J. Saunders, H.T. Siegelmann, R. Kozma, Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to Atari Breakout game, Neural Netw. 120 (2019) 108–115.

[23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI Gym, 2016, arXiv:arXiv:1606.01540.

[24] S.M. Bohte, J.N. Kok, J.A. La Poutré, SpikeProp: backpropagation for networks of spiking neurons, in: ESANN, Vol. 48, Bruges, 2000, pp. 419–424.

[25] S.B. Shrestha, G. Orchard, Slayer: Spike layer error reassignment in time, Adv. Neural Inf. Process. Syst. 31 (2018).

[26] E. Stromatias, M. Soto, T. Serrano-Gotarredona, B. Linares-Barranco, An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data, Front. Neurosci. 11 (2017) 350.

[27] L. Zanatta, F. Barchi, A. Burrello, A. Bartolini, D. Brunelli, A. Acquaviva, Damage detection in structural health monitoring with spiking neural networks, in: 2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT, MetroInd 4.0&IoT, IEEE, 2021, pp. 105–110.

[28] M. Akl, Y. Sandamirskaya, F. Walter, A. Knoll, Porting deep spiking Q-networks to neuromorphic chip loihi, in: International Conference on Neuromorphic Systems 2021, 2021, pp. 1–7.

[29] D. Chen, P. Peng, T. Huang, Y. Tian, Deep reinforcement learning with spiking Q-learning, 2022, arXiv preprint arXiv:2201.09754.

[30] A. Rubino, C. Livanelioglu, N. Qiao, M. Payvand, G. Indiveri, Ultra-low-power FDSOI neural circuits for extreme-edge neuromorphic intelligence, 2020, arXiv:2006.14270.

[31] C. Frenkel, J.-D. Legat, D. Bol, A 28-nm convolutional neuromorphic processor enabling online learning with spike-based retinas, in: 2020 IEEE International Symposium on Circuits and Systems, ISCAS, 2020, pp. 1–5, http://dx.doi.org/10.1109/ISCAS45731.2020.9180440.

[32] F. Barchi, G. Urgese, E. Macii, A. Acquaviva, An efficient mpi implementation for multi-coreneuromorphic platforms, in: 2017 New Generation of CAS, NGCAS, IEEE, 2017, pp. 273–276.

[33] C. Mayr, S. Hoeppner, S. Furber, Spinnaker 2: A 10 million core processor system for brain simulation and machine learning, 2019, arXiv preprint arXiv:1911.02385.

[34] N. Frémaux, H. Sprekeler, W. Gerstner, Reinforcement learning using a continuous time actor-critic framework with spiking neurons, PLoS Comput. Biol. 9 (4) (2013) e1003024.

[35] A. Mahadevuni, P. Li, Navigating mobile robots to target in near shortest time using reinforcement learning with spiking neural networks, in: 2017 International Joint Conference on Neural Networks, IJCNN, IEEE, 2017, pp. 2243–2250.

[36] L. Paulun, A. Wendt, N. Kasabov, A retinotopic spiking neural network system for accurate recognition of moving objects using neucube and dynamic vision sensors, Front. Comput. Neurosci. 12 (2018) 42.

[37] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, S.-C. Liu, Conversion of continuous-valued deep networks to efficient event-driven networks for image classification, Front. Neurosci. 11 (2017) 682.

[38] W. Tan, D. Patel, R. Kozma, Strategy and benchmark for converting deep q-networks to event-driven spiking neural networks, 2020, arXiv preprint arXiv:2009.14456.

[39] S. Shah, D. Dey, C. Lovett, A. Kapoor, AirSim: High-fidelity visual and physical simulation for autonomous vehicles, in: Field and Service Robotics, 2017, URL https://arxiv.org/abs/1705.05065.

[40] Y. Hu, S.-C. Liu, T. Delbruck, v2e: From video frames to realistic DVS events, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 1312–1321.

[41] E.O. Neftci, H. Mostafa, F. Zenke, Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks, IEEE Signal Process. Mag. 36 (6) (2019) 51–63.

[42] S. Hwang, J. Chang, M.-H. Oh, K.K. Min, T. Jang, K. Park, J. Yu, J.-H. Lee, B.-G. Park, Low-latency spiking neural networks using pre-charged membrane potential and delayed evaluation, Front. Neurosci. 15 (2021) 629000.

[43] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, et al., A low power, fully event-based gesture recognition system, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 7243–7252.

[44] M. Fatahi, M. Ahmadi, M. Shahsavari, A. Ahmadi, P. Devienne, evt_MNIST: A spike based version of traditional MNIST, 2016, arXiv preprint arXiv:1604.06751.

[45] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, Commun. ACM 60 (6) (2017) 84–90, http://dx.doi.org/10.1145/3065386.

[46] A. Garofalo, M. Rusci, F. Conti, D. Rossi, L. Benini, PULP-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors, Phil. Trans. R. Soc. A 378 (2164) (2020) 20190155, http://dx.doi.org/10.1098/rsta.2019.0155, arXiv:https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2019.0155.

[47] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.

[48] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 30, No. 1, 2016.

[49] A. Musa, L. Zanatta, F. Barchi, B. Andrea, A. Andrea, A method for accelerated simulations of reinforcement learning tasks of UAVs in AirSim, in: SIMUL 22, 2022.

[50] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al., Isaac gym: High performance gpu-based physics simulation for robot learning, 2021, arXiv preprint arXiv:2108.10470.

[51] Epic Games, Unreal engine, 2019, URL https://www.unrealengine.com.

[52] C.Y. Ho, S.Y. Tseng, C.F. Lai, M.S. Wang, C.J. Chen, A parameter sharing method for reinforcement learning model between airsim and uavs, in: 2018 1st International Cognitive Cities Conference, IC3, IEEE, 2018, pp. 20–23.

[53] D. Gehrig, M. Gehrig, J. Hidalgo-Carrió, D. Scaramuzza, Video to events: Recycling video datasets for event cameras, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 3586–3595.

[54] H. Rebecq, D. Gehrig, D. Scaramuzza, ESIM: an open event camera simulator, in: Conference on Robot Learning, PMLR, 2018, pp. 969–982.

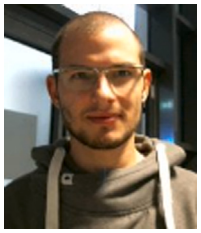
[55] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, D. Scaramuzza, The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM, Int. J. Robot. Res. 36 (2) (2017) 142–149.

[56] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, A. Peste, Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks, 2021, CoRR abs/2102.00554 arXiv:2102.00554.

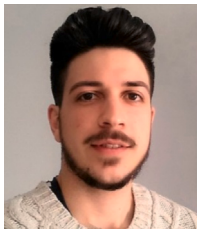
[57] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, L. Benini, A 64-mW DNN-based visual navigation engine for autonomous nano-drones, IEEE Internet Things J. 6 (5) (2019) 8357–8371, http://dx.doi.org/10.1109/JIOT.2019.2917066.

[58] D. Rossi, F. Conti, M. Eggiman, A. Di Mauro, G. Tagliavini, S. Mach, M. Guermandi, A. Pullini, I. Loi, J. Chen, et al., Vega: A ten-core SoC for IoT endnodes with DNN acceleration and cognitive wake-up from MRAM-based state-retentive sleep mode, IEEE J. Solid-State Circuits 57 (1) (2021) 127–139.

[59] Y.-H. Chen, T.-J. Yang, J. Emer, V. Sze, Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices, IEEE J. Emerg. Sel. Top. Circuits Syst. 9 (2) (2019) 292–308, http://dx.doi.org/10.1109/JETCAS.2019.2910232.



Luca Zanatta received the M.Sc. degree in Biomedical Engineering from the Electronics and Telecommunications Department (DET) of Politecnico di Torino in 2019. Since October 2020, he is pursuing a Ph.D. at the ECS Lab of the Alma Mater Studiorum-Università di Bologna, Bologna, Italy. His research focuses on Spiking Neural Networks (SNN) and Reinforcement Learning.



Alfio Di Mauro received the M.Sc. degrees in Electronic Engineering from the Electronics and Telecommunications Department (DET) of Politecnico di Torino in 2016. Since September 2017, he is currently pursuing the Ph.D. at the Integrated System Laboratory (IIS) of the Swiss Federal Institute of Technology of Zurich. His research focuses on the design of digital Ultra-Low Power (ULP) System-on-Chip (SoC) for EventDriven edge computing.

Francesco Barchi received the Ph.D. degree in computer engineering with the Politecnico di Torino, Turin, Italy, in 2020. He is a Research Assistant with the Department of Electrical, Electronic, and Information Engineering, Alma Mater Studiorum-University of Bologna, Bologna, Italy. During his Ph.D., he worked on Spiking Neural Network mapping on the SpiNNaker neuromorphic platform and developed an optimized communication middleware and MPI implementation for the same architecture. His studies focused on deep learning and compilers for heterogeneous architectures. His research interests focus on machine learning, compilers and optimization problems for cyber–physical systems, and heterogeneous architectures.

Andrea Bartolini holds an assistant professor (RTD-B) position at the University of Bologna. He is an active researcher in the domain of power and thermal management in a wide range of computing systems. In this field he has published more than won the Best Paper Award in DATE 2013, the Best IP Paper Award in DATE 2014 Conferences, and the Gauss Award at ISC 2016. He has published more than 85 papers in international peer-reviewed conferences and journals. He has collaborated with several international research and companies. Andrea Bartolini has been the main responsible for the design of advance power management and monitoring support on the first Cavium ThunderX cluster and on the D.A.V.I.D.E. system today ranked on the top20 most energy efficient supercomputers worldwide.

Luca Benini holds the chair of digital Circuits and systems at ETHZ and is Full Professor at the Universita di Bologna. He received a Ph.D. from Stanford University. He has been visiting professor at Stanford University, IMEC, EPFL. He served as chief architect in STmicroelectronics France. Dr. Benini's research interests are in energy-efficient parallel computing systems, smart sensing micro-systems and machine learning hardware. He has published more than 1000 peer-reviewed papers and five books. He is an ERC-advanced grant winner, a Fellow of the IEEE, of the ACM and a member of the Academia Europaea. He is the recipient of the 2016 IEEE CAS Mac Van Valkenburg award,the 2019 IEEE TCAD Donald O. Pederson Best Paper Award and the ACM/IEEE A. Richard Newton Award 2020.

Andrea Acquaviva (Ph.D.) received the Ph.D. degree in Electrical Engineering in 2003 and in Complex Systems for Life Science in 2018. He was research intern at HP Labs Palo Alto, CA, USA (2001–2003) and visiting researcher at LSI lab, EPFL, CH (2007–2008). He has been coordinator of various research projects at European and National level in the field of embedded software, IoT and smart city/energy/buildings. Research interests of Prof. Andrea Acquaviva focus on: (i) compilers and operating systems for heterogeneous embedded platforms, multicores, Internet-of-Things (IoT) devices; (ii) Embedded software for Cyber–Physical Systems (CPS); (iii) Neuromorphic and brain inspired computing. He also carries on research on computational aspects of complex biological systems. Researches by Prof. Andrea Acquaviva (between 2000 and 2022) yielded around 300 papers in international journals and peer-reviewed international conference proceedings.