

PAPER • OPEN ACCESS

A Realistic Simulation Framework for Analog/Digital Neuromorphic Architectures

To cite this article: Fernando M Quintana *et al* 2025 *Neuromorph. Comput. Eng.* **5** 034006

View the [article online](#) for updates and enhancements.

You may also like

- [Gradient-descent hardware-aware training and deployment for mixed-signal neuromorphic processors](#)
Ugurcan Cakal, Maryada, Chenxi Wu et al.
- [Brain-inspired methods for achieving robust computation in heterogeneous mixed-signal neuromorphic processing systems](#)
Dmitrii Zendrikov, Sergio Solinas and Giacomo Indiveri
- [A system design perspective on neuromorphic computer processors](#)
Garrett S Rose, Mst Shamim Ara Shawkat, Adam Z Foshie et al.



PAPER

OPEN ACCESS

RECEIVED
8 October 2024REVISED
7 July 2025ACCEPTED FOR PUBLICATION
14 July 2025PUBLISHED
24 July 2025

Original Content from
this work may be used
under the terms of the
[Creative Commons
Attribution 4.0 licence](#).

Any further distribution
of this work must
maintain attribution to
the author(s) and the title
of the work, journal
citation and DOI.



A Realistic Simulation Framework for Analog/Digital Neuromorphic Architectures

Fernando M Quintana^{1,2,3,*} , Maryada⁴ , Pedro L Galindo³ , Elisa Donati⁴ , Giacomo Indiveri⁴
and Fernando Perez-Peña³

¹ Zernike Institute for Advanced Materials, University of Groningen, Groningen, The Netherlands

² Groningen Cognitive Systems and Materials Center, University of Groningen, Groningen, The Netherlands

³ School of Engineering, University of Cádiz, Puerto Real, Cádiz, Spain

⁴ Institute of Neuroinformatics, University of Zurich and ETH Zurich, Zurich, Switzerland

* Author to whom any correspondence should be addressed.

E-mail: f.m.quintana.velazquez@rug.nl

Keywords: SNN, DPI, neuromorphic, PyTorch, DYNAP-SE

Abstract

Developing dedicated mixed-signal neuromorphic computing systems optimized for real-time sensory-processing in extreme edge-computing applications requires time-consuming design, fabrication, and deployment of full-custom neuromorphic processors. To ensure that initial prototyping efforts exploring the properties of different network architectures and parameter settings lead to realistic results, it is important to use simulation frameworks that match as best as possible the properties of the final hardware. This is particularly challenging for neuromorphic hardware platforms made using mixed-signal analog/digital circuits, due to the variability and noise sensitivity of their components. In this paper, we address this challenge by developing a software spiking neural network simulator explicitly designed to account for the properties of mixed-signal neuromorphic circuits, including device mismatch variability. The simulator, called **A Realistic Simulation Framework for Analog/Digital Neuromorphic Architectures**, is designed to reproduce the dynamics of mixed-signal synapse and neuron electronic circuits with autogradient differentiation for parameter optimization and GPU acceleration. We demonstrate the effectiveness of this approach by matching software simulation results with measurements made from an existing neuromorphic processor. We show how the results obtained provide a reliable estimate of the behavior of the spiking neural network trained in software, once deployed in hardware. This framework enables the development and innovation of new learning rules and processing architectures in neuromorphic embedded systems.

1. Introduction

Mixed-signal neuromorphic circuits emulate the neural and synaptic dynamics observed in real neural systems, sharing similarities such as limited precision, heterogeneity, and high sensitivity to noise [4, 12], which are often overlooked in Artificial Intelligence (AI) workloads. Typically, software simulations of spiking neural network (SNN) use bit-precise activation functions identical for all neurons in a network with high-resolution parameters lacking biologically realistic details. Conversely, tools used in computational neuroscience, such as ‘Neuron’ and ‘NEST’, capture bio-realistic properties of neurons and synapses—such as ion channels and neuronal morphology [7, 9, 23]. In other words, these simulators attempt to replicate real hardware—in this case, *the brain*—but do not offer modules for hyper-parameter optimization. Along similar lines, the neuromorphic community could benefit from a realistic simulator for electronic circuits that emulates the biophysics of neurons and synapses as well as the non-idealities of their analog substrates.

In this paper, we introduce a PyTorch-based [14] simulation platform, **A Realistic Simulation Framework for Analog/Digital Neuromorphic Architectures** (ARCANA), for mixed-signal neuromorphic systems. ARCANA is optimized to simulate neurons and synapses using differential equations derived from their

electronic circuits [1]. This seamless integration provides an easy-to-use infrastructure and libraries for parameter optimization, thereby facilitating the development of neural networks deployable on neuromorphic hardware.

This enables neuromorphic engineers, computational neuroscientists and AI application developers to test and validate SNN architectures in a fast prototyping environment that abstracts the underlying hardware intricacies [10, 15, 18, 22, 26].

A hardware-aware network can be effortlessly deployed on an inference chip, such as those from the dynamic neuromorphic asynchronous processor (DYNAP) family [13, 20]. Moreover, the simulator can be readily adapted to incorporate hardware-specific constraints during model training, enabling a unified framework across various platforms. This approach offers two significant advantages: (1) it simplifies the transfer of a single model across multiple platforms, ensuring seamless cross-platform deployment, and (2) it facilitates benchmarking of different hardware using diverse datasets. Additionally, if a chip supports on-chip learning, deploying a pre-trained network enables continuous adaptation to new real-world data.

2. Methods

The mixed-signal circuit equations used in this framework are based on the differential pair integrator (DPI) circuit [1], which is commonly employed to implement both analog synapse and neuron circuits [4, 16, 21, 25]. The DPI synapse equations faithfully reproduce realistic synaptic dynamics [1] and the corresponding neuron equations mimic the behavior of the adaptive exponential integrate & fire neuron model [2]. Because the DPI is a current-mode log-domain filter, both the synapse and the internal state variables of the neurons are expressed as *currents* (e.g. membrane current), even though theoretical and computational models, typically represent the neuron's state as the membrane *potential*. In the simulator, we adopt terminology and variable names that closely match those used in the electronic circuits.

The mixed-signal neuromorphic processor used to validate the simulations is the DYNAP-SE [13]. It is a multi-core SNN processor with four cores, each with 256 adaptive exponential leaky integrate-and-fire neurons, providing a total of 1024 neurons [13].

2.1. DPI synapse

The simulator models four different types of synapses: two excitatory synapses (AMPA and NMDA) and two inhibitory (GABA_A and GABA_B). They vary in the dynamics and how they interact with the somatic compartment. (1) **AMPA synapses** are composed of a DPI block and are connected to the input of the neuron, modeling biological AMPA receptor; (2) **NMDA synapses**, also an excitatory synapses and linked to neuron's input, however, unlike AMPA synapses, it incorporates a pair of differential blocks to introduce a voltage gating mechanism. This mechanism blocks the synaptic current until the neuron's membrane potential reaching a specific threshold (*Inmda_thr* parameter in ARCANA); (3) **GABA_A, an inhibitory synapses**, implements a current mirror to subtract presynaptic current from the input of the neuron, and (4) **GABA_B synapses** has the current mirror directly connected to the somatic compartment, thus subtracting presynaptic current from membrane current, mimicking a shunting behavior.

The dynamic behavior of the DPI circuit can be approximated as a first-order differential in equation [4]:

$$\tau \frac{d}{dt} I_{\text{syn}} + I_{\text{syn}} = \frac{I_g}{I_r} I_w \quad (1)$$

where $\tau = CU_T/\kappa I_r$ is the synapse decay time constant, C the synapse capacitance, U_T the thermal voltage, κ the transistor subthreshold slope factor, I_r the leakage current, I_g the DPI filter gain, and I_w the base weight current, and I_{syn} the synaptic current. These parameters are tunable and shared among synapses of the same type within each core.

2.2. The DPI neuron

We derived the equations for the neuron model by directly mapping its circuit components. The DPI neuron circuit comprises four main blocks:

- **Input DPI model leak:** the module integrates the DPI inputs from the synapses and the constant DC input current to charge the capacitor C_{mem} , which represents the neuron's leak conductance. It has a series of transistors that control the input current gain, amplitude of DC current, and leakage current that discharges the capacitor. The neuron's input current is composed of the constant DC current plus the synaptic currents aggregated from *AMPA* and *NMDA*, as well as *GABA_A*. Additionally, the *GABA_B* synapse is connected to the leakage current, directly discharging the capacitor.

- **After-hyperpolarization (AHP) block:** this block provides slow negative feedback that models spike frequency adaptation. When a postsynaptic spike occurs, it integrates the event into a recurrent negative (AHP) current, which is subtracted from the input, effectively suppressing the activity of the neuron.
- **Positive feedback and spike generation block:** this block mimics activation and inactivation of sodium channels through a positive feedback circuit. When the neuron current starts to ramp up, the current driving the inverter circuits is fed back into the capacitor C_{mem} , further increasing the neuron's membrane current. Consequently, the I_{mem} current grows exponentially until the inverter circuits finish switching, at which point a spike is generated and the reset block is activated.
- **Reset block:** this block mimics the potassium channels. After a spike event, it resets the membrane current by creating a short circuit to ground, which discharges the neuron's membrane capacitance and directs the membrane current I_{mem} to ground. This discharge period, controlled by a bias parameter, prevents C_{mem} from recharging, resulting in an absolute reset and a refractory period. Once this period ends, the I_{mem} current recharges the capacitor C_{mem} and the neuron resumes integrating its input spikes.

The dynamics of these modules are governed by the following differential equations [4]:

$$\left(1 + \frac{I_g}{I_{\text{mem}}}\right) \tau \frac{dI_{\text{mem}}}{dt} + I_{\text{mem}} \left(1 + \frac{I_{\text{ahp}}}{I_\tau}\right) = I_\infty + f(I_{\text{mem}}) \quad (2a)$$

$$\tau_{\text{ahp}} \frac{dI_{\text{ahp}}}{dt} + I_{\text{ahp}} = I_{\text{ahp}\infty} u(t) \quad (2b)$$

$$I_\infty = \frac{I_g}{I_\tau} (I_{\text{in}} - I_{\text{ahp}} - I_\tau) \quad (2c)$$

where I_{mem} denotes the subthreshold membrane current. I_{ahp} is the AHP current responsible of spike-frequency adaptation. I_∞ is the maximum current a neuron would reach asymptotically. I_τ represents leakage current of the neuron, τ is the neuron time constant, and I_{in} is the total input current from both synaptic sources and the constant DC input. All these parameters can be configured both on DYNAP-SE chip as well as the simulator.

The term $f(I_{\text{mem}})$ in equation (2a) represents the positive feedback current as described in section 2.2, and it is well modeled by an exponential function. [4]:

$$f(I_{\text{mem}}) = \frac{I_{fb}}{I_\tau} (I_{\text{mem}} - I_g) \quad I_{fb} = \frac{I_0^{\frac{1}{\kappa+1}} I_{\text{mem}}^{\frac{\kappa}{\kappa+1}}}{1 + e^{-\alpha(I_{\text{mem}} - I_{\text{th}})}} \quad (3)$$

where α and I_{th} are hyper-parameters, I_0 is dark current and κ is transistor slope factor.

2.2.1. Parameters:

The digital-to-analog (DAC) bias-generator circuits configure bias parameters by generating specific magnitudes of current to govern neuronal and synaptic properties, including time constant, refractory period, and synaptic weights [5]. Internally, these parameters are represented as ‘coarse’ and a ‘fine’ value, where *coarse* $\in [0, 7]$ and *fine* $\in [0, 255]$. These parameters exhibit variability, also known as mismatch, due to fabrication non-idealities, which inherently limits user control (see section 3). Moreover, the assigned values are non-observable and cannot be directly retrieved from the circuits, so we rely on oscilloscope traces for measurement. The bias parameters that can be set on simulation are listed in table 1. In the case of I_{th} and α they are fitting parameters of the positive feedback and not a direct bias in DYNAP-SE.

2.3. Calibration

We performed a one-time calibration (per chip) to align the simulation parameters (in pA) with those on the chip (in a.u.) using incremental adjustments. We primary calibrated the membrane leak time constant, I_τ , because it is the only bias that directly influences the membrane current — a quantity measurable by an oscilloscope. All other parameters were kept fixed to ensure consistency during the calibration process.

Similarly, we also evaluated the above-threshold neural response by increasing the DC input (I_{DC}) and neuron gain (I_{th}). A slight adjustment of the leak current ($I_\tau = 4.1\text{pA}$) was sufficient to match the simulation and silicon traces, reducing the neuron's decay time constant. This calibration resulted in the values: $I_{\text{th}} = 500\text{pA}$, $I_\tau = 4.1\text{pA}$, and $I_{\text{DC}} = 36.6\text{pA}$ for simulation. These current values are obtained by setting a [coarse, fine] pair that controls the on-chip DAC current-bias generator [5]: $I_{\text{th}} = [6, 88]$, $I_\tau = [6, 22]$, and $I_\tau = [2, 57]$. Figure 1(a) shows a close match between the simulation traces (dashed orange line) and the silicon neuron traces (solid blue line), confirming that the simulation equations accurately capture the parameters and that their influence on silicon neuron behavior is faithfully modeled.

Table 1. ARCANA model parameter and DYNAP-SE biases correspondence.

ARCANA	DYNAP-SE bias	Description
Igain_mem	IF_THR_N	Neuron gain
Itau_mem	IF_TAU1_N	Membrane time constant
refractory ^a	IF_RFR_N	Refractory period
Idc	IF_DC_P	DC input
Itau_ampa	NPDP1E_TAU_F_P	AMPA synaptic time constant
Igain_ampa	NPDP1E_THR_F_P	AMPA synaptic gain
Iw_ampa	PS_WEIGHT_EXC_F_N	AMPA synaptic base weight current
Inmda_thr	IF_NMDA_N	NMDA voltage gating
Itau_nmda	NPDP1E_TAU_S_P	NMDA synaptic time constant
Igain_nmda	NPDP1E_THR_S_P	NMDA synaptic gain
Iw_nmda	PS_WEIGHT_EXC_S_N	NMDA synaptic base weight current
Itau_gabaa	NPDP1I_TAU_F_P	GABA _A synaptic time constant
Igain_gabaa	NPDP1I_THR_F_P	GABA _A synaptic gain
Iw_gabaa	PS_WEIGHT_INH_F_N	GABA _A synaptic base weight current
Itau_gabab	NPDP1I_TAU_S_P	GABA _B synaptic time constant
Igain_gabab	NPDP1I_THR_S_P	GABA _B synaptic gain
Iw_gabab	PS_WEIGHT_INH_S_N	GABA _B synaptic base weight current

^a The refractory period in ARCANA is in milliseconds compared to the bias current in DYNAP-SE.

Following a similar approach, we calibrated the synapse parameters by recording the neuron's membrane current in response to a single spike. For the AMPA synapse, the membrane current trace (figure 1(b)) was obtained with $I_\tau = 8\text{pA}$ ($[coarse = 2, fine = 20]$), $I_{th} = 6.5\text{pA}$ ($[coarse = 2, fine = 20]$) and $I_w = 50\text{pA}$ ($[coarse = 5, fine = 99]$). For the GABA_A inhibitory synapse (figure 1(c)), we set $I_\tau = 20\text{pA}$ ($[coarse = 2, fine = 80]$), $I_{th} = 350\text{pA}$ ($[coarse = 4, fine = 80]$), $I_{DC} = 6.5\text{pA}$ ($[coarse = 2, fine = 10]$) and $I_w = 400\text{pA}$ ($[coarse = 5, fine = 200]$). Note that for inhibitory synapse, the neuron's membrane current must be above its resting level to observe the inhibitory effect.

We observed a higher deviation in traces for AMPA ($MSE = 1.888 \times 10^{-7}$) and GABA_A synapses ($MSE = 1.879 \times 10^{-6}$) compared to the traces with constant input current. This discrepancy may have arisen from the scale of the simulated voltage. When the input is provided via a synapse, the somatic voltage only reaches about 80 mV, whereas with constant DC current, it can rise to 800 mV due to a positive feedback mechanism. Another possible explanation is that parasitic capacitance from the circuit layout and fabrication, which is not accounted for in the simulation, could contribute to the difference.

Here, we calibrated a single neuron such that all others remain within a 20% margin of variability caused by mismatch. A more rigorous approach would involve a statistical analysis of the entire population, as described in [27] using chip-in-loop calibration, but this is beyond the scope of the present work.

This calibration process is integrated in subsequent experiments (see sections 4.2 and 4.3). These experiments serve as validation tests, where a network trained in simulation was deployed on the chip and produced outcomes similar to those observed in ARCANA.

3. Hardware mismatch

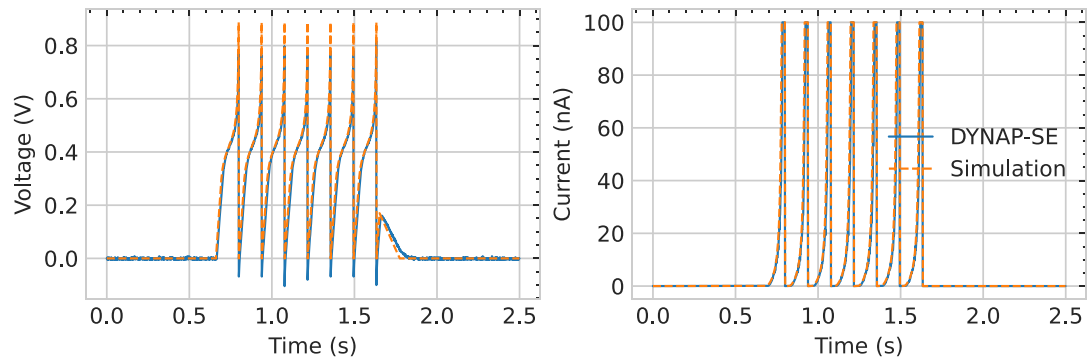
Inherent variability in the analog substrate can cause computational discrepancies relative to idealized simulations, imposing constraints that are beyond the user's control. Typically, the coefficient of variation for all parameters ranges from 10%–20% [27].

This mismatch is evident while observing from membrane leak. Figure 2 shows the distribution of time constant (τ_{mem}) computed from membrane traces with bias settings applied via IF_TAU1_N. The coarse bias is fixed at 5, while the fine bias is swept from 74 to 190. Notably, as the τ value is modified, its standard deviation decreases proportionally.

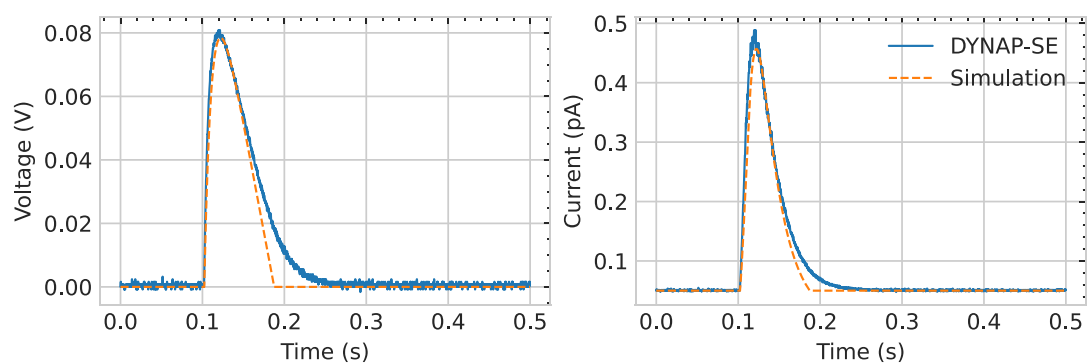
ARCANA incorporates mismatch in both neuron and synapse models, allowing users to assess their models' resilience to parameter variability. Users can select the desired degree of variability, starting with deterministic simulations for initial testing and gradually introducing mismatch as needed.

4. Results

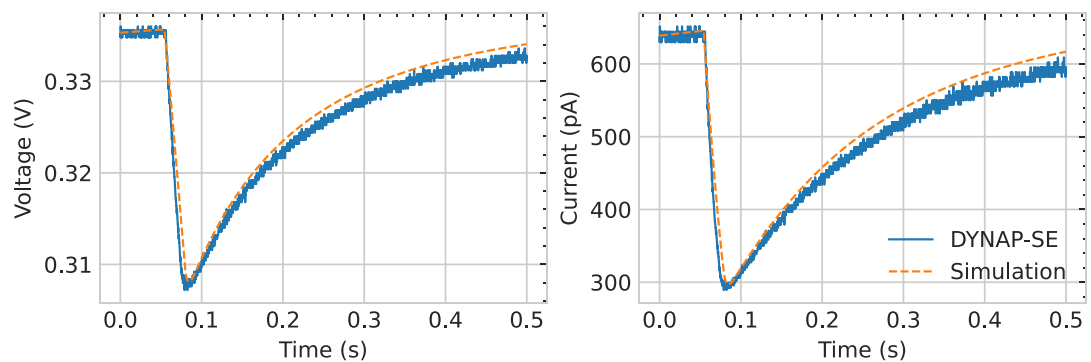
We present three experiments that assess the dynamics and accuracy of our PyTorch-based implementation of the DPI neuron and synapse model. The first experiment, described in section 4.1, features a frequency resonator used solely to validate the training process in simulation; it was not deployed on the chip. In all



(a) Using a constant DC input, we compared ARCANA simulator with the DYNAP-SE chip. For visualization purposes, the current was capped at 100 nA, although in practice it would rise exponentially until the firing threshold was reached.



(b) Comparison between ARCANA and DYNAP-SE chip using a AMPA input synapse. The traces exhibit a mean squared error (MSE) of 1.888×10^{-7} .



(c) Comparison between ARCANA and DYNAP-SE chip using a GABA_a input synapse. The traces exhibit a mean squared error (MSE) of 1.879×10^{-6} .

Figure 1. ARCANA and DYNAP-SE comparison, where on each figure, the *left pane* represents the voltage and the *right pane* the soma current.

experiments, we employed the *autograd* tool to automatically calculate gradients, thereby facilitating the tuning and optimization of complex models.

Once we validated the training process, we trained a small network to perform binary classification using backpropagation through time. The trained network was then deployed on DYNAP-SE chip (see section 4.2). In the last experiment (section 4.3), we present a proof-of-concept for event-based three-factor local plasticity (ETLP) [17]. In this experiment, a network was trained in simulation using ETLP to distinguish between two spike patterns. After training, we deployed it on the DYNAP-SE chip for inference.

These experiments serve as validation for the algorithm's compatibility with hardware-aware training. Deploying pre-learned weights and synaptic connections on mixed-signal hardware paves the way for innovative real-world applications in neuromorphic computing. Notably, before deployment, it is critical to

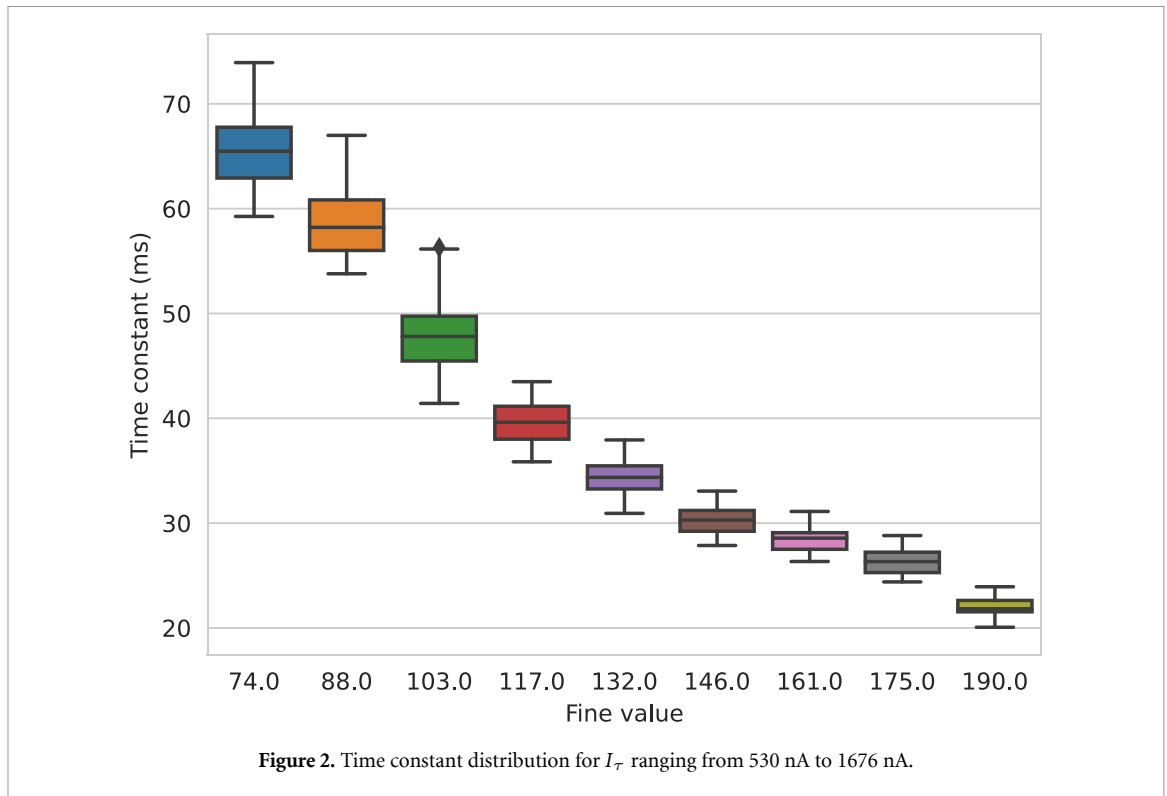


Table 2. Parameters used for spike frequency resonator experiment.

Parameter	Value
Initial neuron I_τ	4×10^{-12}
Initial neuron I_g	20×10^{-12}
I_{DC}	10×10^{-12}
Positive feedback gain (I_g)	20×10^{-12}
Positive feedback normalization (α)	2.0×10^9
Learning rate	5×10^{-3}
Optimizer	Adam

perform a one-time manual calibration per chip, as detailed in section 2. In future work, this calibration process can be automated using autograd and a chip-in-loop paradigm [11].

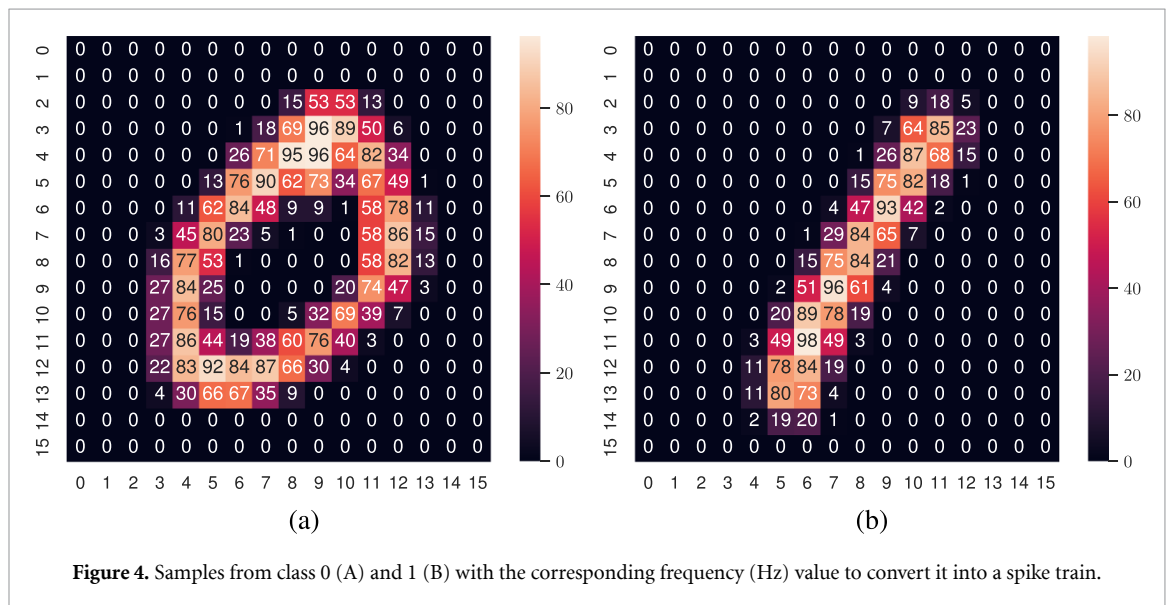
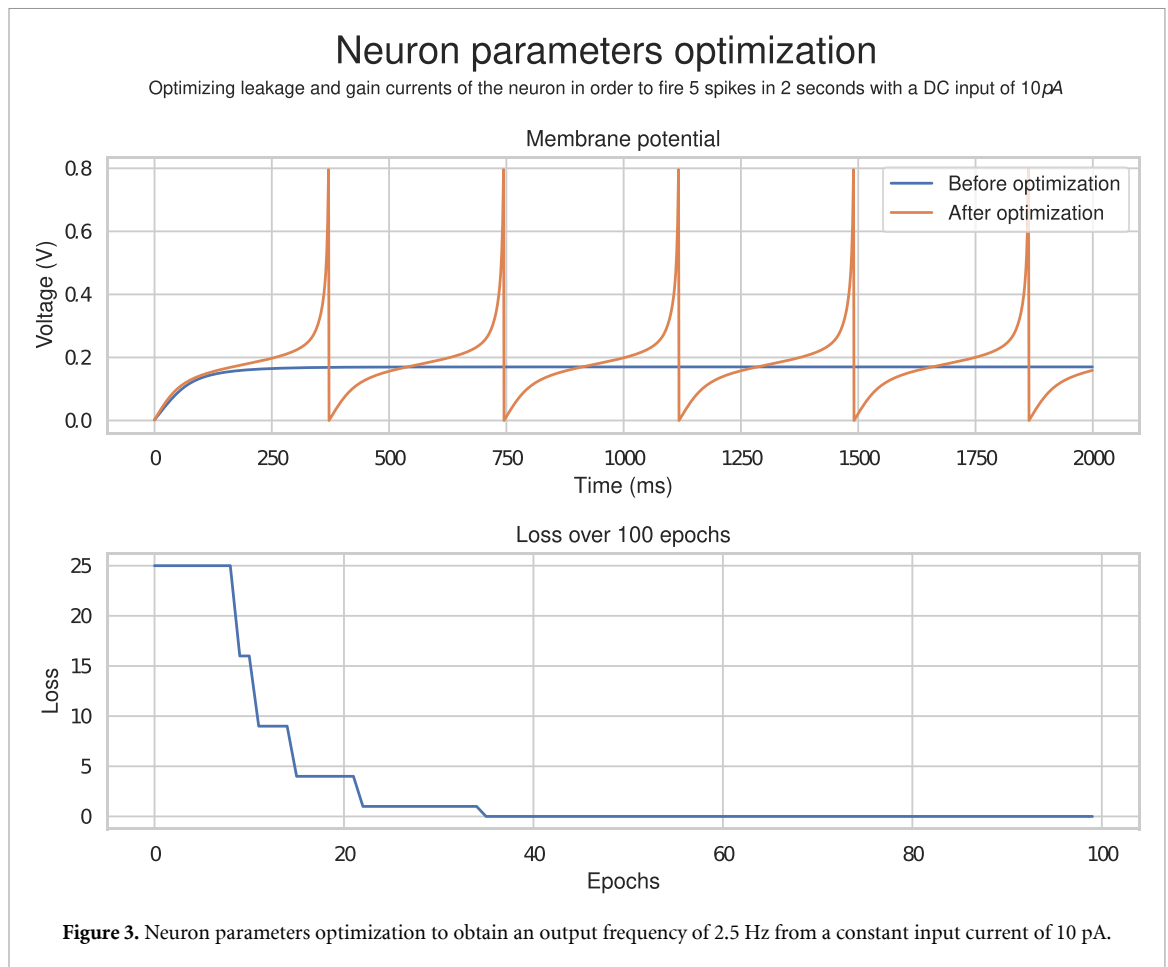
4.1. Spike frequency resonator

In this experiment, we perform a basic task to showcase the use of the *autograd* tool to optimize neuron parameters. The task involves constant current injection to a neuron with $I_{DC} = 10$ pA. The objective is to modify neuron gain (I_{th}) and membrane time constant (I_τ) to achieve the desired firing frequency of 2.5 Hz. Figure 3(top) shows the initial response of the neuron to DC injection (solid blue line). The absence of a spike response indicates the need to adjust the neuron's parameters. Instead of manually calibrating these parameters, we use autograd to automatically determine the optimal values so that the neuron fires at the user-specified frequency of 2.5 HZ.

The loss function is the mean squared error between the number of output spikes and the target frequency. This loss is calculated with respect to the neuron parameters that we aim to optimize— neuron gain (I_{th}) and membrane time constant (I_τ). Figure 3(bottom) illustrates that the training was completed (loss $\simeq 0$) in fewer than 40 epochs. Figure 3(top) also shows neuron's response post-optimization in orange. The parameters used for this experiment are listed on table 2.

4.2. Binary image classification

In this experiment, we trained the network to perform image classification for digit recognition. For simplicity, we only consider 0 and 1 of the MNIST dataset. We trained the connection matrix of the network and subsequently deployed the trained network on DYNAP-SE for on-chip inference. The aim is to ensure that the designated neuron on the chip exhibits a higher firing rate than the baseline for its preferred digit (0



or 1). The network receives the pixel intensity encoded as firing rate with a Poisson distribution as shown in figure 4. The parameters used for this experiment are listed on table 3.

4.2.1. Syncing ARCANA and DYNAP-SE:

To ensure a faithful deployment of a trained network, it is essential to match the leakage and gain currents for both neurons and synapses on the chip and the ARCANA simulator, as described in section 2.3. On the DYNAP-SE chip, the base weight for each synapse type can be tuned using bias currents, similar to other neuron and synaptic parameters (see section 2.2). This base weight is common to all neurons in a single core. Hence, to modify the weights for a specific neuron, we alter the number of synapses connecting two neurons

Table 3. Parameters used for binary classification problem.

Parameter	Value
Neuron I_τ	1.8×10^{-12}
Neuron I_g	45×10^{-12}
I_{DC}	240×10^{-12}
Positive feedback gain (I_g)	225×10^{-12}
Positive feedback normalization (α)	1.0×10^9
GABA _A /AMPA synapse I_τ	4×10^{-12}
GABA _A /AMPA synapse I_g	10×10^{-12}
GABA _A /AMPA synapse base weight	400×10^{-12}
Learning rate	1×10^{-3}
Optimizer	SGD
Maximum number of input synapses	40

(connection matrix). This necessitates the quantization of the learned connection matrix into integer values. We dealt with quantization constraint by employing quantization-aware training (QAT) procedure.

4.2.2. Network training:

The classification task discussed here is linearly separable, therefore, we train a single-layer network with 256 input channels and 2 readout neurons, with one neuron representing digit 0 and the other representing digit 1. Each neuron can receive input with multiple excitatory (AMPA) and inhibitory (GABA_A) synapses. We initialized AMPA and GABA_A connection matrices with a uniform distribution. We pruned the synapses to meet the fan-in limitations (64 per neuron) of the chip. We introduced mismatch in the synaptic weights bias (I_w) during training. However, all other neuron and synaptic parameters were manually calibrated (3) and remain unchanged during training. Note that the framework also supports incorporating mismatch across all parameters during training, even though this feature is not utilized in this experiment.

Training with 32-bit floating-point weights provides a broader dynamic range but complicates on-chip deployment, as DYNAP-SE uses quantized weights. With the help of quantization techniques such as QAT [29], we reduced the precision of weights from float to integer, resulting in efficient computation while maintaining high accuracy during inference on the chip. A rounding operator was used to map the floating point tensor to a quantized representation:

$$x_q = \text{round}(x_f).$$

During training, QAT introduces a ‘mock’ low precision in the forward pass, while the backward pass remains full precision (figure 5(a)). To deal with the quantization operation gradient, the straight-through estimator surrogate gradient is used. This approach allows the gradient to be transmitted unaltered through the fake-quantization operator.

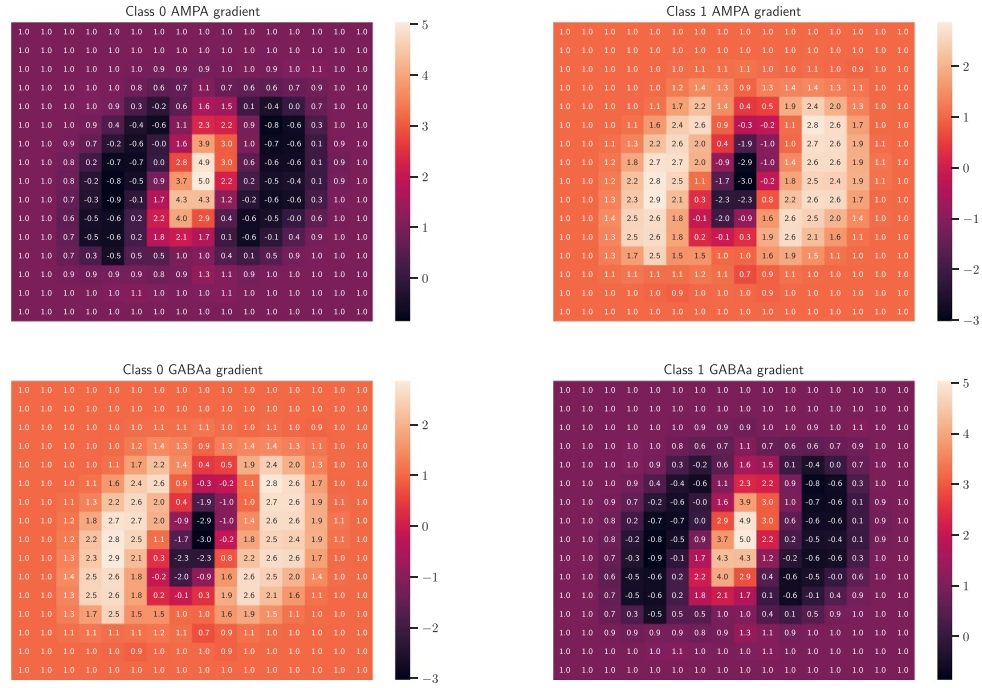
As previously stated, the DYNAP-SE chip also restricts the fan-in of a neuron, permitting only 64 input synapses per neuron. This restriction can differ for different chips, however it must be taken in account during training to avoid exceeding the synapse limit and risking a topology that is incompatible with the hardware. To address this challenge, we penalize the model for higher fan-in by adding L1 and/or L2 regularization terms. For instance, for W as the connection matrix and fan_in as the desired number of input synapses (here fan_in = 64), we used a L1 regularization, where $L1 = \lambda \sum |W - \text{fan_in}|$. λ is a hyperparameter that controls the strength of the regularization. If the sum of the rounded weights is not yet exactly equal to the desired value, a final adjustment can be made by adding a constant value C to each weight, defined as $C = (\text{fan_in} - \sum(\text{round}(W)))/N$. Here N is the number of input neurons, and the sum is over all the input weights of each postsynaptic neuron.

4.2.3. Loss:

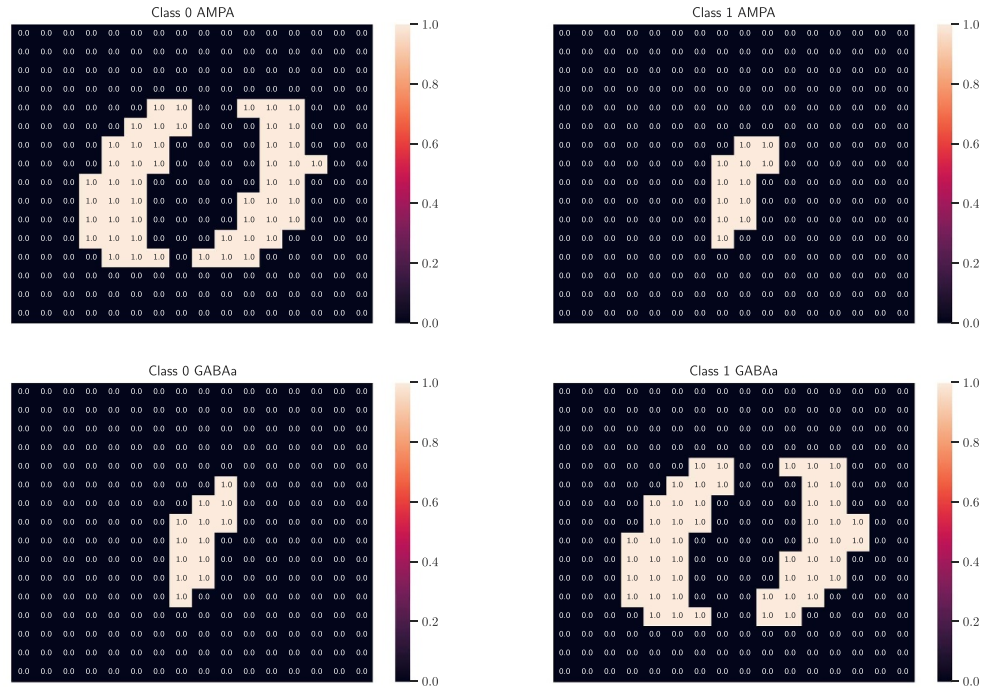
The loss function for this task is a softmax cross-entropy applied to the neuron inputs. Here, the network is optimized to increase the input current of the output neuron corresponding to the presented class while reducing the current of the other neuron. We defined the loss function on the total input current of each neuron (difference between the AMPA and GABA_A current) in equation (4a).

$$L = - \sum_{n=1}^N \sum_{q=1}^Q y_{n,q} \log \frac{e^{x_{n,q}}}{\sum_{i=1}^Q e^{x_{n,i}}} \quad (4a)$$

$$x_n = \sum_{t=0}^T I_{\text{AMPA}}(t)_n - I_{\text{GABA}_A}(t)_n \quad (4b)$$



(a) AMPA and GABA_A synapses gradients during the training process for classes 0 and 1. The value of the gradients are stored and calculated in full-precision.

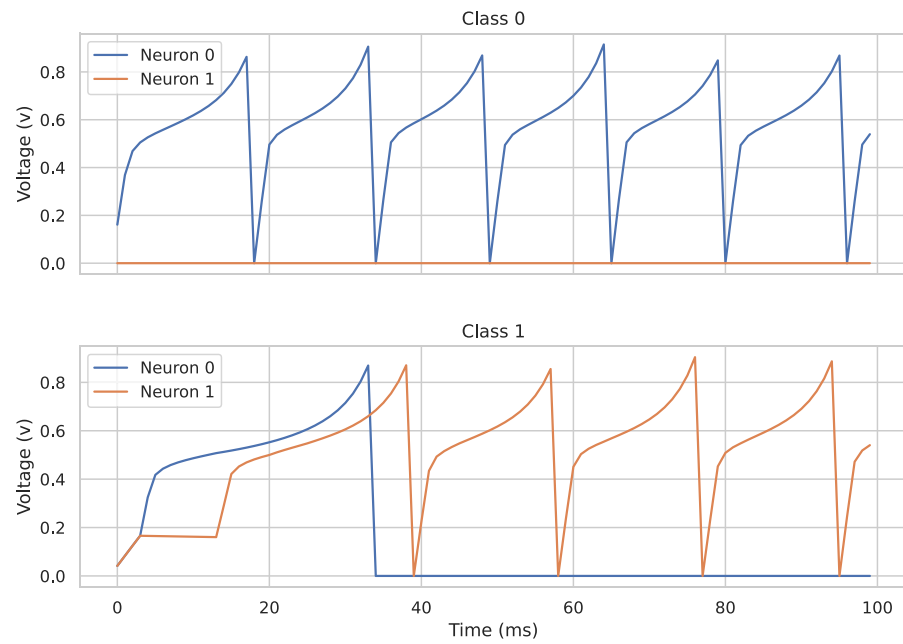


(b) AMPA and GABA_A final connection matrix for class 0 and 1 obtained after training and quantizing them.

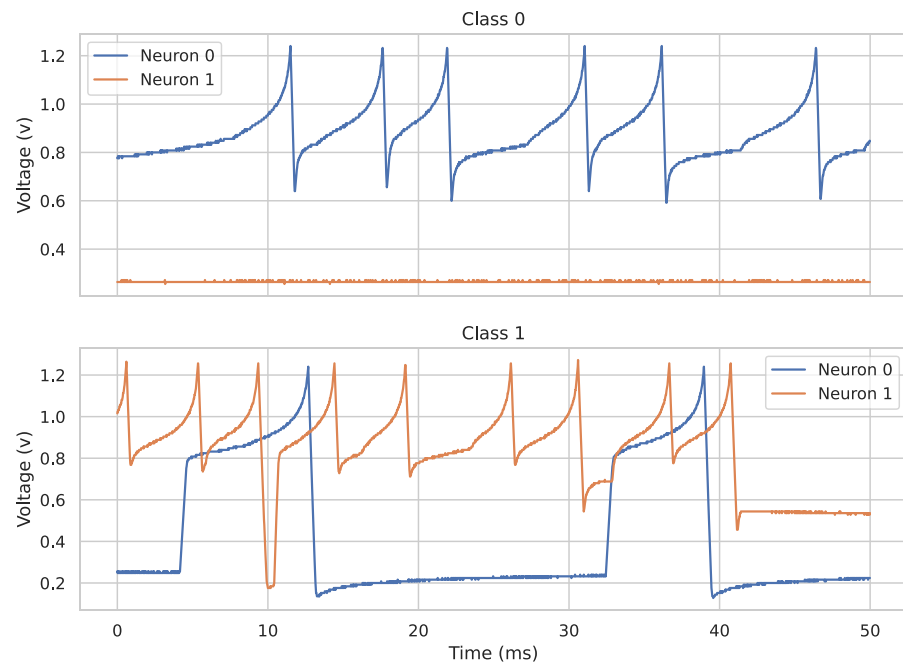
Figure 5. AMPA and GABA_A gradients and final quantized weight matrices.

where $y_{n,q}$ is the target class probability for sample n and class q , x_n is the difference between the accumulation over time of the AMPA and GABA_A currents over time, Q the number of classes, N the number of samples, and T the total simulation time.

The final connection matrix obtained after training is then rounded, as shown in figure 5(b). Before deploying the network on the DYNAP-SE, we performed an inference mode evaluation in simulation



(a) Output neurons voltages simulated in ARCANA when receiving a sample from class 0 and class 1.

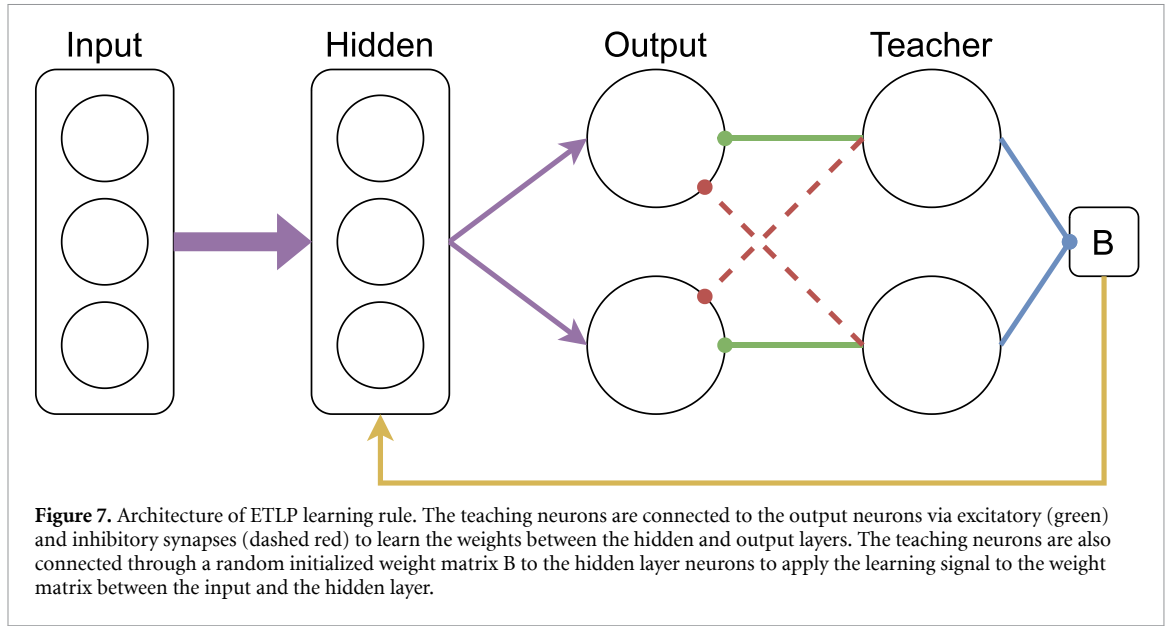


(b) Output neurons voltages recorded on DYNAP-SE when receiving a sample from class 0 and class 1. The figure shows the output of the neuron voltage recorded directly from the hardware, from a sample of each class.

Figure 6. Output neuron voltages for classes 0 and 1 recorded in simulation (a) and on chip (b).

(figure 6(a)). As anticipated, the neurons spike behavior only when exposed to their respective selective digits.

The network was deployed on the hardware as a last step in this process. During on-chip inference, 2115 test samples of MNIST digits (0 or 1) were randomly selected and presented to the network for 50 ms each, interleaved with 50 ms rest periods with no stimulus. The firing response of the neurons was recorded for each presentation. Figure 6(b) shows representative traces recorded on the chip during one inference session. Running entirely on hardware, the emulated network achieved an accuracy of 99.11% in predicting the



stimulus. This demonstrates the feasibility of training network connections in software and efficiently deploying it on a mixed-signal chip.

4.3. Learning rules for DYNAP-SE

In this experiment, we showed how ARCANA is a promising tool-chain for end-to-end network training and on-chip inference post-deployment. An important advantage of having a simulator for a mixed-signal processor such as DYNAP-SE is the possibility of testing various learning rules such as ETLP [17] with the neuron and synapse model implemented on the hardware. Thus, it serves as a proof-of-concept to test the viability of these learning algorithms for specific hardware before designing the circuit layout of the learning rule for on-chip implementation.

In this experiment, we trained a network using ETLP [17] and deployed the final weights on DYNAP-SE. ETLP is a supervised three-factor local learning rule composed of (1) a presynaptic trace, (2) a postsynaptic value, and (3) an external teaching signal from teacher neurons that triggers the learning (figure 7). The weight update follows equation (5b),

$$\tau_{\epsilon} \frac{d\epsilon}{dt} = \epsilon + x \quad (5a)$$

$$\Delta W = \sigma'(v(t)) \epsilon(t) I(t) \quad (5b)$$

where ΔW is the weight change, ϵ is the pre-synaptic trace computed by accumulating the input spikes, $\sigma'(v) = (1 + \alpha|v|)^{-2}$ is a non-linear function depending of the soma voltage $v(t)$ [28], x the input pre-synaptic spike, τ_{ϵ} the pre-synaptic trace time constant and $I(t)$ the input from the teaching neurons.

In this experiment, we trained a network (as illustrated in 7) to distinguish between two input frequencies —1 Hz and 10 Hz per channel. This system can be applied to anomaly detection. The network architecture consists of 50 input neurons, 50 hidden neurons, and 50 output neurons, along with 2 teacher neurons. A high-frequency teacher signal at 50 Hz facilitates learning of the preferred class, while a low-frequency teacher at 5 Hz simulates noise. Synaptic weight changes follow the ETLP rule. For inference, the trained weights were deployed on the DYNAP-SE chip. The parameters used on this experiment are listed on table 4.

Figure 8 shows spike activity of the whole network during the training and inference process in simulation. The output neurons fire independently for the initial 4 s of the pattern presentation. In the next 20 s, both tasks are presented alternatively in the network by increasing the firing rate of the input neurons from each class. This allows the hidden and output layers to learn the relation between the classes and the input pattern. During inference, the teacher neurons are set to a low-rate firing state (5 Hz), a subset of neurons in output layer exhibit higher firing response for their preferred class, validating correct association was learned during training.

4.3.1. On-chip inference:

once the network was trained and validated for inference in simulation, the trained weights were mapped into a network created on DYNAP-SE chip without any preprocessing step. Figure 9 shows the spike activity

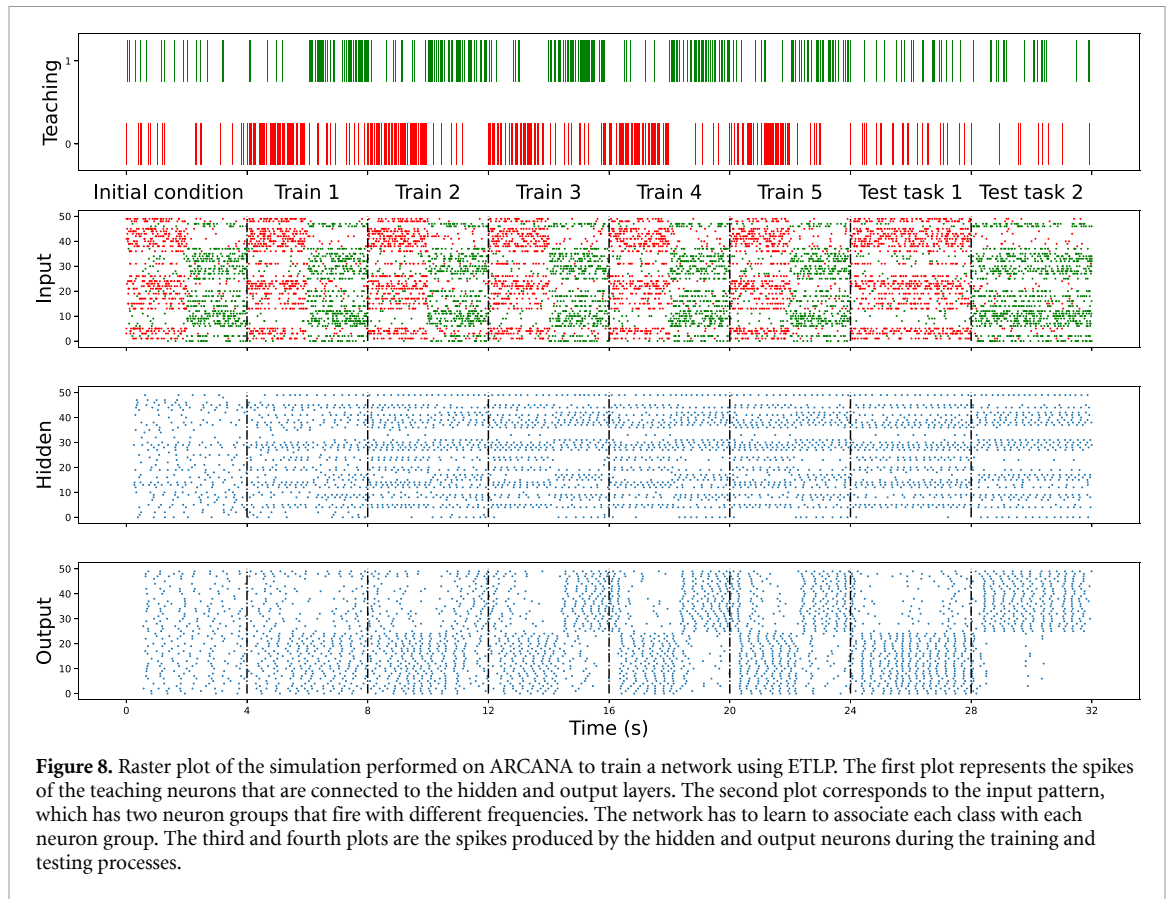


Figure 8. Raster plot of the simulation performed on ARCANA to train a network using ETLF. The first plot represents the spikes of the teaching neurons that are connected to the hidden and output layers. The second plot corresponds to the input pattern, which has two neuron groups that fire with different frequencies. The network has to learn to associate each class with each neuron group. The third and fourth plots are the spikes produced by the hidden and output neurons during the training and testing processes.

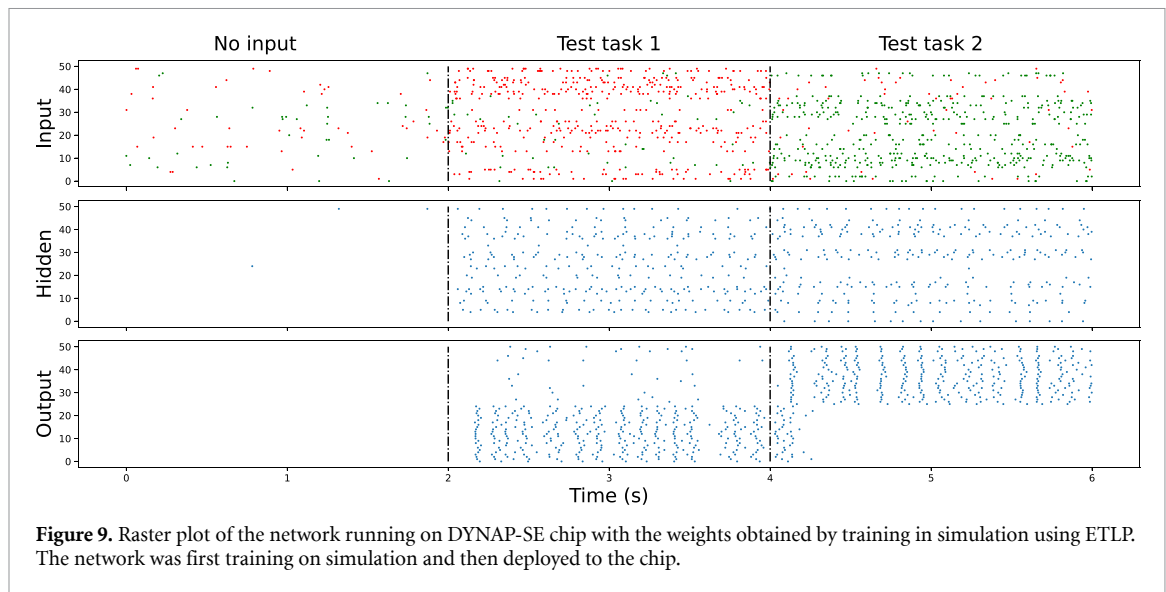
Table 4. Parameters used for ETLF experiment. The presynaptic trace time constant is the same as the neuron time constant.

Parameter	Value
Neuron I_{τ}	4.32×10^{-12}
Neuron I_g	20×10^{-12}
I_{DC}	0
Positive feedback gain (I_g)	225×10^{-12}
Positive feedback normalization (α)	1.0×10^9
GABA _a /AMPA synapse I_{τ}	4×10^{-12}
GABA _a /AMPA synapse I_g	10×10^{-12}
GABA _a /AMPA synapse base weight	200×10^{-12}
Maximum number of input synapses	40

of the whole network on chip, exhibiting the desired behavior for each individual input. During on-chip inference, with no teacher provided, the bottom raster plot shows two distinct neuron groups, each exhibiting increased activity when their preferred pattern is presented.

5. Discussion

In this paper, we presented the ARCANA simulator for mixed-signal hardware and validated its performance using the DYNAP-SE chip. The advantage of ARCANA is its ability to use of the PyTorch ‘autograd’ feature, which allows the optimization of the internal parameters of the network. While other similar simulators developed in parallel to this work have been proposed within the Rockpool framework [3], specifically tailored for DYNAP-SE2 chip [20], ARCANA boasts a more versatile application and is generic to any processor incorporating DPI based neuron models, independent of the specific simulation framework used. This adaptability of ARCANA is achieved by fine-tuning parameters such as the positive-feedback exponential function parameters and the chip constant values, including the transistor slope factor and neuron capacitor. Furthermore, ARCANA allows us to optimize not only the weights but other parameters as well. In addition, ARCANA includes the possibility to apply mismatch into the neuron and synapse parameters found in the hardware. This mismatch can be included in the training framework, making the



system more robust to variability in the biases. It also provides a more realistic environment for simulation, where the non-idealities of the hardware are taken into account.

Similar effort has been done for digital neuromorphic systems, for instance *Lava* software framework for Loihi 2 and sPyNNaker for SpiNNaker [19]. ARCANA is also capable of simulating circuit-level neuronal behavior using frameworks like Neural Engineering Framework (NEF). [6, 24] (a methodology for building large-scale neural models of cognition). NEF has previously been used to model analogue circuits [8]. Integrating ARCANA with NEF could allow converting the physical model of the DYNAP-SE hardware into a symbolic representation. Finally, we demonstrated the feasibility of implementing ETLP in hardware and applied it to online training in mixed-signal hardware. This opens new doors to the development of new local learning rules in embedded systems that go beyond the ones presented so far.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: <https://github.com/ferqui/ARCANA/tree/main/docs/examples/trainingRecord>.

Acknowledgments

F M Q was supported by FPU Grant (FPU18/04321) from the Spanish Ministry of Universities. This work is supported by the HORIZON EUROPE EIC Pathfinder Grant ELEGANCE (Grant No. 101161114), and has received funding from Swiss National Science Foundation (SNSF 200021E_222393). This work has received funding from the Swiss State Secretariat for Education, Research and Innovation (SERI).

Conflict of interests

The authors declare no competing interests.

Code availability

<https://github.com/ferqui/ARCANA>

ORCID iDs

Fernando M Quintana 0000-0001-5042-9399

Maryada 0009-0009-9706-5989

Pedro L Galindo 0000-0003-0892-8113

Elisa Donati 0000-0002-8091-1298

Giacomo Indiveri 0000-0002-7109-1689

Fernando Perez-Peña 0000-0003-3586-2930

References

- [1] Bartolozzi C and Indiveri G 2007 Synaptic dynamics in analog VLSI *Neural Comput.* **19** 2581–603
- [2] Brette R and Gerstner W 2005 Adaptive exponential integrate-and-fire model as an effective description of neuronal activity *J. Neurophysiol.* **94** 3637–42
- [3] Cakal U, Maryada C W, Ulusoy I and Muir D R 2024 Gradient-descent hardware-aware training and deployment for mixed-signal neuromorphic processors *Neuromorph. Comput. Eng.* **4** 014011
- [4] Chicca E, Stefanini F, Bartolozzi C and Indiveri G 2014 Neuromorphic electronic circuits for building autonomous cognitive systems *Proc. IEEE* **102** 1367–88
- [5] Delbruck T and Van Schaik A 2005 Bias current generators with wide dynamic range *Analog Integr. Circuits Signal Process.* **43** 247–68
- [6] Eliasmith C and Anderson C H 2003 *Neural Engineering: Computation, Representation and Dynamics in Neurobiological Systems* (MIT Press)
- [7] Gewaltig M-O and Diesmann M 2007 NEST (neural simulation tool) *Scholarpedia* **2** 1430
- [8] Hazan A and Tsur E E 2022 Neuromorphic neural engineering framework- inspired online continuous learning with analog circuitry *Appl. Sci.* **12** 4528
- [9] Hines M and Carnevale N 1997 The NEURON simulation environment *Neural Comput.* **9** 1179–209
- [10] Lammie C and Azghadi M R 2020 Memtorch: a simulation framework for deep memristive cross-bar architectures *2020 IEEE Int. Symp. on Circuits and Systems (ISCAS)* pp 1–5
- [11] Maryada S S-M, Sorbaro M, Laje R, Buonomano D V and Indiveri G 2023 Stable recurrent dynamics in heterogeneous neuromorphic computing systems using excitatory and inhibitory plasticity
- [12] Mead C 2023 Neuromorphic engineering: in memory of misha mahowald *Neural Comput.* **35** 343–83
- [13] Moradi S, Qiao N, Stefanini F and Indiveri G 2018 A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs) *IEEE Trans. Biomed. Circuits Syst.* **12** 106–22
- [14] Paszke A et al 2019 Pytorch: an imperative style, high-performance deep learning library *Advances in Neural Information Processing Systems* vol 32 (Curran Associates, Inc.) pp 8024–35 (available at: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>)
- [15] Pehle C, Billaudelle S, Cramer B, Kaiser J, Schreiber K, Stradmann Y, Weis J, Leibfried A, Müller E and Schemmel J 2022 The brainscales-2 accelerated neuromorphic system with hybrid plasticity *Front. Neurosci.* **16** 795876
- [16] Quan J, Liu Z, Li B and Luo J 2023 Ultra-low-power compact neuron circuit with tunable spiking frequency and high robustness in 22 nm FDSOI *Electronics* **12** 2648
- [17] Quintana F M, Perez-Peña F, Galindo P L, Neftci E O, Chicca E and Khacef L 2024 ETLF: event-based three-factor local plasticity for online learning with neuromorphic hardware *Neuromorph. Comput. Eng.* **4** 034006
- [18] Rasch M J, Moreda D, Gokmen T, Gallo M L, Carta F, Goldberg C, Maghraoui K E, Sebastian A and Narayanan V 2021 A flexible and fast pytorch toolkit for simulating training and inference on analog crossbar arrays *2021 IEEE 3rd Int. Conf. on Artificial Intelligence Circuits and Systems (AICAS)* pp 1–4
- [19] Rhodes O et al 2018 sPyNNaker: a software package for running PyNN simulations on SpiNNaker *Front. Neurosci.* **12** 2018
- [20] Richter O, Wu C, Whatley A M, Köstinger G, Nielsen C, Qiao N and Indiveri G 2024 DYNAP-SE2: a scalable multi-core dynamic neuromorphic asynchronous spiking neural network processor *Neuromorph. Comput. Eng.* **4** 014003
- [21] Rubino A, Livanelioglu C, Qiao N, Payvand M and Indiveri G 2020 Ultra-low-power FDSOI neural circuits for extreme-edge neuromorphic intelligence *IEEE Trans. Circuits Syst. I* **68** 45–56
- [22] Spilger P et al 2020 Hxtorch: pytorch for brainscales-2: perceptrons on analog neuromorphic hardware *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge and Mobile for Embedded Machine Learning: Second Int. Workshop, IoT Streams 2020 and First Int. Workshop, (ITEM 2020, Co-Located With ECML/PKDD 2020, Ghent, Belgium, 14–18 September 2020, Revised Selected Papers 2)* (Springer) pp 189–200
- [23] Stimberg M, Brette R and Goodman D F 2019 Brian 2, an intuitive and efficient neural simulator *eLife* **8** e47314
- [24] Tsur E E 2021 *Neuromorphic Engineering: the Scientist's, Algorithms Designer's and Computer Architect's Perspectives on Brain-Inspired Computing* (CRC Press)
- [25] Vuppunuthala S and Pasupureddi V S 2023 3.6-pj/spike, 30-hz silicon neuron circuit in 0.5- μ m, 65 nm CMOS for spiking neural networks *IEEE Trans. Circuits Syst. II*
- [26] Yu Z, Menzel S, Strachan J P and Neftci E 2024 Integration of physics-derived memristor models with machine learning frameworks (arXiv:2403.06746)
- [27] Zendrikov D, Solinas S and Indiveri G 2023 Brain-inspired methods for achieving robust computation in heterogeneous mixed-signal neuromorphic processing systems *Neuromorph. Comput. Eng.* **3** 034002
- [28] Zenke F and Ganguli S 2018 Superspike: supervised learning in multilayer spiking neural networks *Neural Comput.* **30** 1514–41
- [29] Zmora N, Wu H and Rodge J 2021 Achieving fp32 accuracy for int8 inference using quantization aware training with tensorrt (available at: <https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt/>)