# End-to-End Analog Edge AI Architecture For Scalable Energy-Efficient On-Chip Learning

Mohamed Watfa, Gilles Sassatelli, Alberto Garcia-Ortiz

Preprint: Under Review

# End-to-End Analog Edge AI Architecture For Scalable Energy-Efficient On-Chip Learning

Mohamed Watfa*†, Alberto Garcia-Ortiz† (*Senior, IEEE*), Gilles Sassatelli* (*Fellow, IEEE*)
*LIRMM, University of Montpellier, CNRS, Montpellier, France
†ITEM, University of Bremen, Bremen, Germany
{mohamed.watfa, sassatelli}@lirmm.fr, agarcia@item.uni-bremen.de

***Abstract*—Equilibrium Propagation (EP) has emerged as a promising method in deep learning that leverages analog processing and memristive devices for efficient deep learning. However, practical challenges such as reduced accuracy in voltage variation calculations, non-ideal characteristics of memristive devices, and analog processing blocks complicate its application. Existing EP models, often idealized, overlook these issues, making them impractical for silicon-based implementations. Moreover, current implementations mainly serve as proof-of-concept architectures, falling short in solving complex problems effectively. Our study introduces and conducts a comprehensive analysis of various EP circuit implementations to identify efficient solutions for on-device training. We explore different formulations of update rules and propose hardware-aware gradient quantization and accumulation for batch training. Our findings show that simple update rules in current analog implementations fail on modest problems, but analog-compatible modifications can achieve performance comparable to ideal models. Furthermore, we propose a viable architecture for developing analog/mixed-signal systems capable of end-to-end training, thereby paving the way towards practical and efficient analog machine learning solutions.**

***Index Terms*—analog neural networks, hardware optimization, memristive technology, equilibrium propagation, spice simulation**

## I. Introduction

Deep learning is being increasingly used to address complex problems across diverse domains [1], [2]. However, this progress entails significant energy consumption due to the high computational demands [3]. These issues are critical in the context of edge devices, which operate under stringent power and computational constraints [4]. Furthermore, there's a growing need for on-device learning [5], which not only improves the efficiency and real-time responsiveness of these devices [6], [7], but also allows them to adapt to changing data environments [8], an essential feature for maintaining accuracy over time. However, achieving this goal is hampered by the inherent complexities of implementing neural networks in hardware, notably in optimizing for both inference and on-device learning.

To address the first concern, a promising line of research has been the exploration of memristive crossbars to perform the matrix-vector multiplication in the analog domain [9]. These circuits address the memory access and data movement problem in digital accelerators [10] and boast significant improvements in performance and energy consumption over state-of-the-art digital designs [11]. Nonetheless, the integration of these technologies in mixed-signal accelerators is often limited by the need for data converters between layers, introducing substantial power and area overheads [12], [13], [14]. Moreover, when memristors are used in purely inference mixed-signal accelerators, finding the optimal weights is not trivial: since training is often done offline, a naive linear mapping from matrix values to crossbar conductances can lead to low accuracy [15].

In spite of this, the use of memristors in purely analog accelerators presents a compelling case. Analog accelerators, combined with an analog-friendly learning algorithm, such as the Equilibrium Propagation (EP) algorithm [16], can potentially leverage the unique properties of memristors, such as their continuous resistance states and their inherent non-linearity, to perform the computations directly in the analog domain, circumventing the need for energy-intensive digital-to-analog and analog-to-digital converters. This approach could not only lead to significant improvements in energy efficiency and computational density, but also paves the way towards the realization of efficient on-chip learning.

The EP algorithm is one of the many methods that have been explored to address the challenges of implementing Backpropagation in hardware [17], [18], [19], such as non-locality of information exchange and the requirement for different kinds of computations in the forward and backward phases of training [20]. The EP algorithm, explained graphically in Fig. 1, is a two-step optimization algorithm based on the principle of energy minimization. It defines the energy function as $F_{\boldsymbol{\theta}} = E_{\boldsymbol{\theta}} + \beta \cdot C$, where $\boldsymbol{\theta}$ is the adjustable parameter of the system, $E_{\boldsymbol{\theta}}$ is the internal energy, and $\beta$ is a parameter that modulates how the external force, $C$, influences the internal state $\boldsymbol{s}$ of the system. By defining the energy function this way and by associating a state variable $s$ to each neuron, EP requires the same kind of computation for both steps of the optimization problem (called *free* and *nudging* phases), potentially easing hardware implementation.

Nevertheless, the gap between the theoretical allure of EP and its practical application remains wide. This gap stems from several factors, including the challenges of accurately calculating voltage variations, the non-ideal behaviors of memristive devices, and the intricacies of analog implementations which, to date, have been largely experimental and limited in their ability to solve complex problems effectively. Our work seeks to bridge this divide by making following contributions:

- **Comprehensive Analysis of EP Implementations**: We introduce and conduct a rigorous analysis of various EP

circuit implementations, identifying the parameters that are crucial for enhancing the performance and reliability for on-device training.
- **Innovative Hardware-Aware Modifications**: We present hardware-aware gradient quantization and accumulation methods for batch training. This includes the development of a novel circuit that implements these methods in hardware, achieving performance levels comparable to ideal models and thus advancing the state-of-the-art.
- **End-to-End Architecture**: We design and present a robust architecture for developing analog/mixed-signal systems capable of end-to-end training, laying the groundwork for practical and highly efficient analog machine learning solutions.

This paper is organized as follows: In Sec. II, we present our proposed hypothetical architecture optimized for hardware implementation of EP. In Sec. III, we detail the methodology used in our experimental analysis. In Sec. IV, we present our findings on four key aspects: the impact of batching in EP, the impact of various EP learning rules, the benefits of gradient quantization, and the necessity of adapting the update rule to the unique characteristics of memristors. In Sec. V, we discuss the hardware implications of our results and compare our method to the current state-of-the-art analog-based solutions. Finally, we conclude and offer perspectives for future work.

## II. End-to-End Mixed-Signal Architecture For On-Chip Learning

A high-level floorplan of our proposed architecture is shown in Fig. 2a. It primarily comprises of crossbar arrays connected through bidirectional analog blocks. Each crossbar array is associated with a Programming Unit (Prog. Unit) and a Row Select Unit (Row Select), which are only active during the training phase. High-dimensional input vectors are transferred into the chip and deserialized via a shift register (Shifter). The input is then converted from digital to analog via a digital-to-analog (D/A) interface to enable analog processing in the crossbar arrays. The processed output is subsequently retrieved through an analog-to-digital (A/D) interface. Unlike typical implementations where data converters are used between layers, incurring significant energy and hardware costs, our design uses such converters only at the system's interface.

The architecture integrates dual crossbars: one for storing analog weights via memristors and another for storing analog gradients via capacitors during training. For efficient training, every memristor is paired with its own gradient-storing capacitor. However, in extreme resource-constrained settings, the architecture could be adapted to selectively update a few randomly-selected memristors at a time, integrating the capacitors within the programming units to save space. Each crossbar is further composed of two sub-crossbars to enable the representation of negative weights, a necessary adaptation for devices like memristors that only store positive values. The crossbars are also equipped with row and column multiplexers (MUX) to facilitate access to the programming elements during training. These are effectively transparent during inference.

The basic structure of a memristor crossbar that can be used to implement a fully-connected layer in EP is shown in Fig. 2e. Designs for incorporating other layers have also been proposed [9]. The array operates in two modes controlled by the state of the multiplexers. In normal mode, voltages are applied to all rows simultaneously as pulses with durations long enough for the circuit to reach equilibrium ①. In programming mode, conductance updates can be done individually, row-wise, or column-wise, depending on the requirement. For instance, as demonstrated in the figure, all elements in row $i$ can be updated simultaneously by applying appropriate programming voltages to the column buffers and grounding row $i$, while other rows remain disconnected ②.

Unlike neural networks that process memristive crossbars digitally on a layer-by-layer basis, EP requires simultaneous equilibrium across all layers. As a result, all layers are interconnected with bidirectional buffers and non-linearities instead of expensive A/D and D/A converters. While the requirement for the entire system to achieve equilibrium simultaneously precludes the possibility of reusing layers, this limitation is not unique to our system; it is a common characteristic of most in-memory analog neural networks that use memristive crossbars for weight storage. Despite these constraints, the representational capacity of equilibrium-based networks, even with only a few layers, is quite high, rivaling that of multi-layer recurrent neural networks [21].

On-chip training can be supported through two approaches. In the first approach, each memristor is equipped with its own programming unit. This has the advantage of supporting true local learning and extremely fast processing times as all the memristors could, in theory, be processed in parallel. However, this comes at the cost of larger chip area and high power consumption.

The alternative, depicted in Fig. 2, positions the programming units outside the crossbar arrays, allowing them to be shared by multiple memristors. Moreover, using fewer but more precise programming units is crucial for realistic implementations, as it allows for better detection of small voltage variations across training phases. While this approach reduces hardware costs, it comes at the price of slower processing speeds and the need for complex logic to select individual cells. To enhance speeds, programming could occur simultaneously across entire rows or columns by duplicating programming units according to row or column elements. The former approach is depicted in Fig. 2.

The programming units have two functions: they compute and accumulate gradients for the memristors, and generate the necessary voltage to update them.

- **Gradient Computation**: This occurs in two stages. During the free phase, the voltage across the memristor is sampled and stored using a sample-and-hold (SAH) block. This voltage is then processed according to the EP update rule. The same process is repeated during the nudging phase. The voltages from both phases are subtracted to produce the loss gradient for a given sample. If batching is used, this new gradient is added to the previously stored one in the capacitor, and then saved back into it.

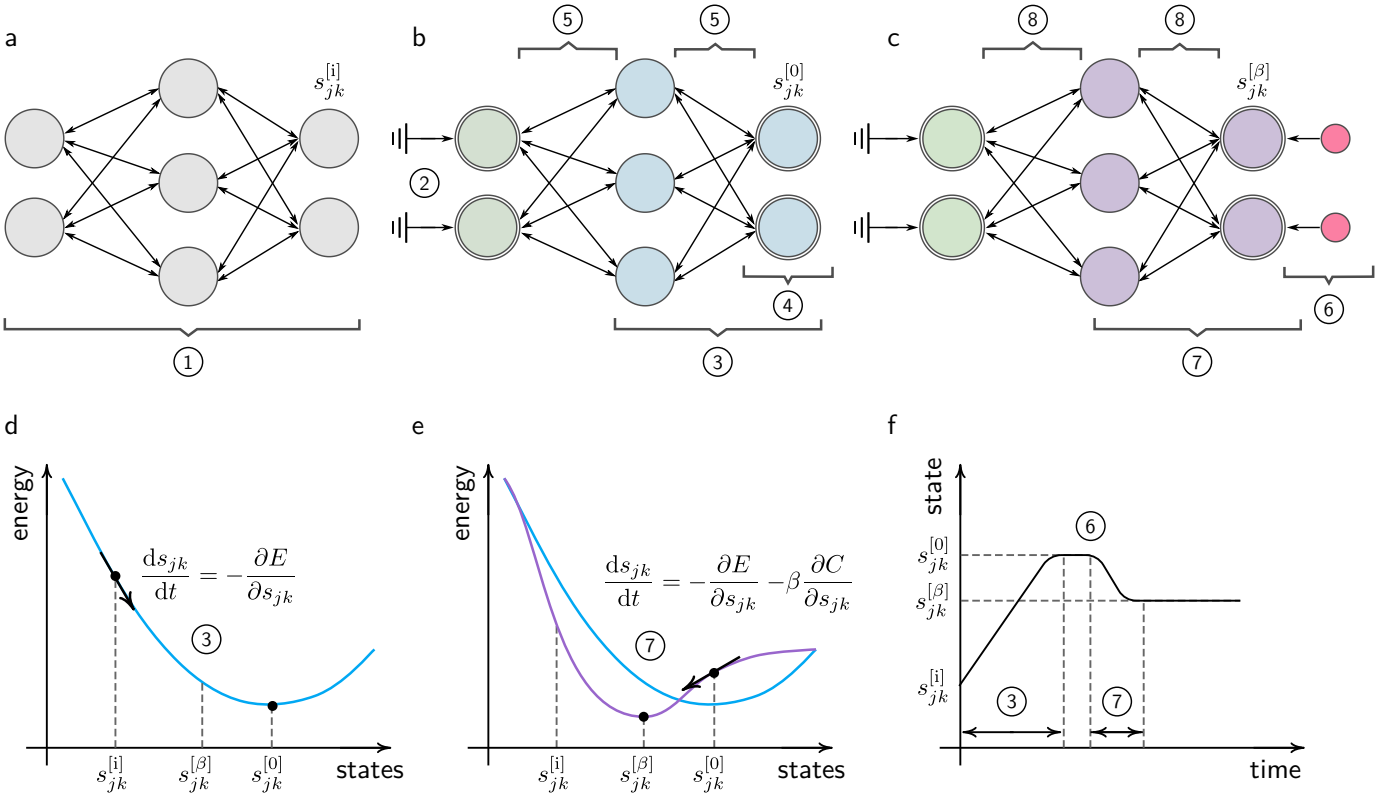



Fig. 1: Training in EP: (**a**) shows a network with interconnected nodes. Unlike conventional neural networks, the links between nodes are bidirectional, as shown by the double-headed arrows. This is because EP is an equilibrium based algorithm where each neuron exchanges information with its neighbors. Every node is associated with a state $s$, collectively forming the state vector $\boldsymbol{s}$. The first step involves initializing the state vector to zero ①. (**b**) Before training begins, some nodes are designated either as inputs or outputs, as indicated by double strokes. At the start of training (or *free* phase), input nodes are clamped to the input vector $\boldsymbol{x}$ ②. The remaining nodes are allowed to dynamically evolve according to equation $\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t} = -\frac{\partial E}{\partial \boldsymbol{s}}$ until they settle at a free equilibrium state, $\boldsymbol{s}^{[f]}$ ③. At the end of the free phase, the network's prediction can be derived from the output nodes ④. The gradient of the energy, $E_{\boldsymbol{\theta}}$, of the system with respect to the parameter $\theta_{ij}$ is a function of only the states it is connected to, $s_i$ and $s_j$. For the entire network, this term is denoted as $\left(\frac{\partial E}{\partial \boldsymbol{\theta}}\right)^{[f]}$ ⑤ (**c**) With the input nodes still clamped to the input vector, the output nodes are pushed to the target vector's value, $\boldsymbol{y}$, using loss nodes shown in red ⑥. The remaining unclamped nodes are allowed to once again evolve dynamically following the equation $\frac{\mathrm{d}\boldsymbol{s}}{\mathrm{d}t} = -\frac{\partial E}{\partial \boldsymbol{s}} - \beta \cdot \frac{\partial C}{\partial \boldsymbol{s}}$, where $C$ is the cost function, until they reach a new equilibrium, $\boldsymbol{s}^{[n]}$, which is better than $\boldsymbol{s}^{[f]}$ in terms of the prediction error ⑦. The gradient of the energy, $F_{\boldsymbol{\theta}} = E_{\boldsymbol{\theta}} + \beta \cdot C$, of the system with respect to the parameter vector $\boldsymbol{\theta}$ is denoted as $\left(\frac{\partial F}{\partial \boldsymbol{\theta}}\right)^{[n]}$ ⑧. Finally, the parameter vector is updated in proportion to $-\frac{1}{\beta}\left[\left(\frac{\partial F}{\partial \boldsymbol{\theta}}\right)^{[n]} - \left(\frac{\partial E}{\partial \boldsymbol{\theta}}\right)^{[f]}\right]$.

- **Memristor Programming**: The memristor can either be updated using the newly calculated gradient or by preloaded values from an external storage. This data is then converted into a programming pulse depending on the chosen scheme. The pulse driver uses this voltage to drive the column line to program the memristor.

The computation of the gradient according to the EP rule requires the evaluation of the circuit at two equilibrium points. The first equilibrium point is measured at the end of the free phase during which the output units are left floating. At the end of this phase, the outputs, representing the prediction of the circuit, are measured and compared with the desired outputs. A loss current, representing the gradient of the loss function with respect to the output, is then injected into the output nodes. This current propagates through the circuit until it reaches a new equilibrium state.

The computation of the loss current can be done using either analog or digital methods. For instance, the gradient of the mean-squared error (MSE) can be easily realized in pure analog hardware using a subtractor (opamp) and a voltage-controlled current source. However, the gradient of loss functions that require more complex calculations, such as the crossentropy loss, are more easily done in digital, necessitating A/D and D/A interfaces. This digital processing is only needed at the output layer and is essential for interfacing with the external world, despite the additional overhead.

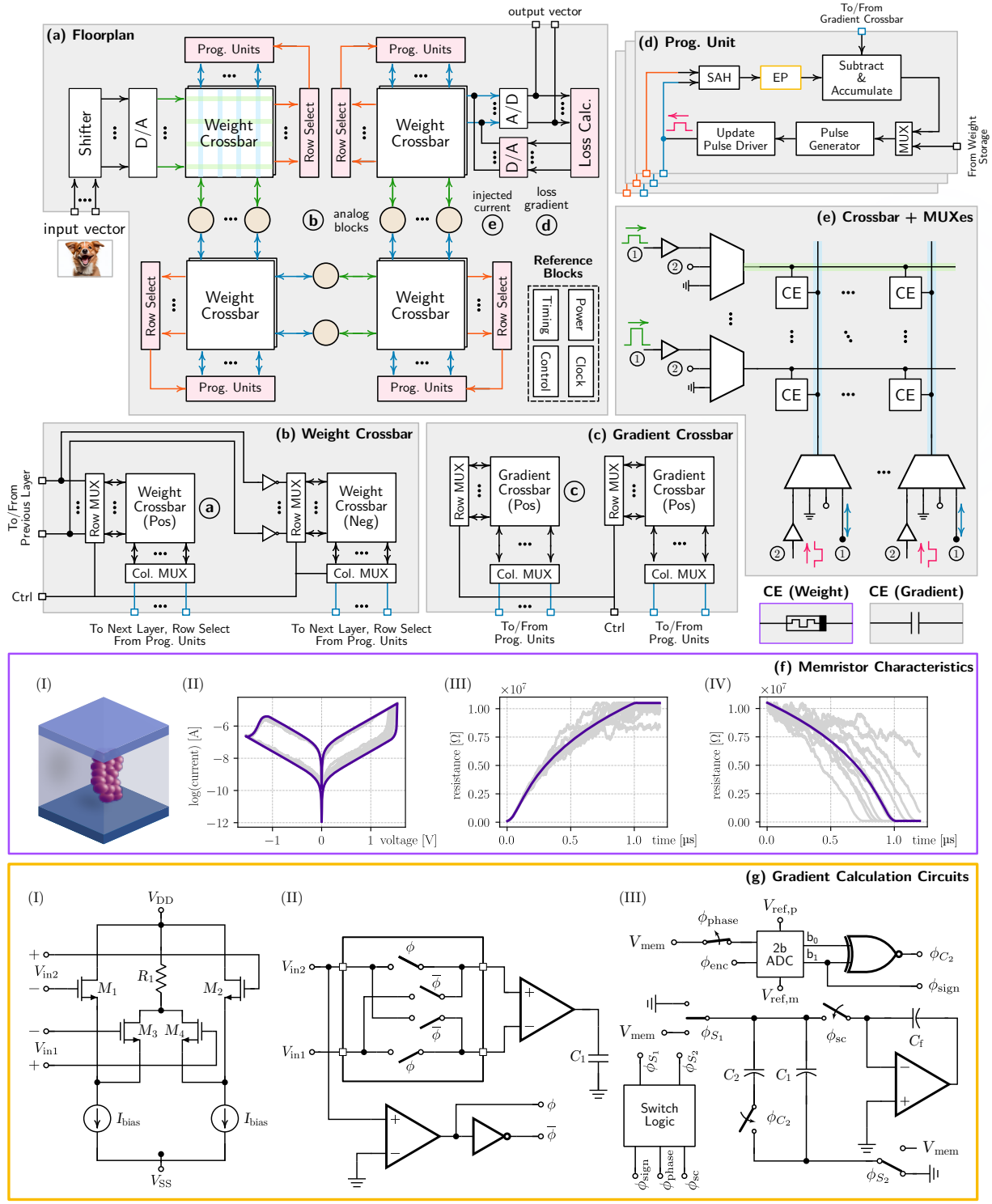



Fig. 2: Hardware architecture for implementing the EP algorithm. The D/A converters transform the input into a vector of voltages, $V$. These are applied to the rows (shown in green) of the first weight crossbar, which is connected to a series of other crossbars via analog blocks ⓑ, which, as depicted in Fig. 3, could include components such as diodes to introduce nonlinearity to the network. Applying a voltage $V$ to a column of conductances $G_j$ produces a current given by the dot product of $V - V_j$ and $G_j$, where $V_j$ is the steady-state voltage of column $j$. These steady-state voltages, depend not only on the inputs, but on the entire circuit. They are the result of the minimization of the energy function inherent to the whole circuit, as required by EP. This corresponds to step ③ in Fig. 1, concluding the free phase. If training the network is desired, the voltage across each conductance (given by the difference between the row and column voltages after steady-state) should first be stored in a capacitor ⓒ ⑤. The network's output is then compared against the true output using a suitable loss function ⓓ. The resulting losses are injected into the output nodes ⓔ ⑤, causing a perturbation that travels back through the network ⑦, altering the voltages at all nodes ⑧. Using the EP update rule, these new set of voltages are used to calculate the loss gradients.

## III. Methodology

### A. Framework for Conducting Experiments

This study conducts four experiments. The first explores the necessity of batching in hardware implementations, a contrast to the non-batching approach used in [22]. The second experiment evaluates the effectiveness of simplified EP update rules, noting that [22] limited their evaluation to the absolute value function on the MNIST dataset. The third experiment examines how compression affects learning capabilities, focusing on gradient quantization. For these experiments, an ideal linear resistor models the weight. In the fourth experiment, this linear resistor is replaced with a memristor model to assess if learning is feasible with simplified rules or if adaptation to the memristor's characteristics is necessary. Finally, we evaluate a simple memristor programming scheme.

Given the complexity of the full architecture and the breadth of experiments involved, simulating every aspect in SPICE proved extremely time-consuming. To optimize this process, the simulations were divided into two parts: the free and nudging phases of the EP algorithm were conducted in SPICE to accurately reflect the hardware behavior. The other parts of the process, such as sensing, gradient computation, and weight updating, were carried out in Python, significantly accelerating the process. To streamline the construction of analog neural networks in a manner similar to using Keras[1], we used a tool specifically developed for this purpose, called EBANA (Energy-Based Analog Neural Network Framework) [23]. This framework leverages Python's flexibility to easily model the effects of different levels of accuracy, including the implementation of complex models like memristors. The procedure is illustrated in Fig. 3.

TABLE I: Training parameters of conducted experiments. The abbreviations used are: DS for dataset, LN for the number of nodes in a layer, LR for learning rate, MSE for mean-squared error, and CE for crossentropy.

| DS | LN | LR | Loss | Optim. |
|---|---|---|---|---|
| Iris | 9–10–6 | $10^{-9}$ – $10^{-10}$ | MSE | SGD |
| MNIST | 1569–100–20 | $10^{-8}$ – $10^{-8}$ | CE | Adam |

For all experiments, the network architecture consisted of an input layer, a hidden layer, and an output layer. The specific number of nodes and other training parameters are indicated in Table I. Initial learning rates for the hidden and output layers were set to those in the table, and then repeated for nearby values to reduce the bias of bad learning rates. Due to the time-intensive nature of analog simulations, the MNIST dataset was limited to 1000 samples to ensure the experiments could be completed in a reasonable timeframe. The training accuracy for Iris was evaluated using the entire dataset, whereas a subset of the traning dataset was used for MNIST. Finally, each experiment varied only one parameter at a time to clearly assess its effect on the training dynamics.

All experiments were conducted for the same number of epochs. Following this, the data was filtered to assess the impact of the selected variables on model accuracy. Only the configuration with the best learning rates were chosen. Subsequently, the filtered results were used to make box and whisker plots with three axis: accuracy, epoch, and updates. An epoch indicates the network has processed the entire dataset once, linking directly to training time. Updates refer to how often the network's parameters are changed, which depends on both the epoch count and batch size. In setups using memristors, updates also relate to programming time, power use, and inversely, to hardware lifespan. Each point on the boxplots represents the outcome of a single experiment, highlighting the best accuracy achieved along with the epoch and update count that reached this level. While this does not capture the differences between stable and oscillating networks, closer examination of the data reveals that oscillating networks tend to, on average, perform worse than stable networks, a trend implicitly reflected in the figures.

In the batching experiment, for the Iris dataset, batch sizes were varied from 1 to 105, incrementing by factors of 105. "Small batch" refers to groupings of batches sized 3, 5, and 7, while "large batch" includes sizes of 15, 35, and 105. For the MNIST dataset, experiments were limited to batch sizes of 1, 10, and 40, representing "no batching", "small batch", and "large batch", respectively.

### B. Memristor Modeling

Memristors represent an emerging category of memory technologies that distinguishes itself by relying on resistance change rather than charge storage to store information. Unlike conventional resistors, memristors do not adhere strictly to Ohm's Law. To understand how the unique properties of memristors affect the training of neural networks, we integrated a memristor model developed by researchers at Stanford into the EBANA simulation tool [24]. This model, based on metal-oxide RRAM technology, captures the dynamic behavior of memristors, including DC cycling, pulsed operation, and the inherent stochastic variability associated with resistive switching. It incorporates factors such as electric field influence, temperature-enhanced ion migration, and local temperature effects due to Joule heating, allowing for a realistic simulation of memristor behavior.

Given EBANA's design as an open-source analog neural network simulator, the direct use of Verilog-A models (which is how the Stanford memristor model is distributed) poses a compatibility issue, primarily because there are no freely available Verilog-A compilers[2]. To bridge this gap, the memristor model was rewritten in Python with minor modifications to ensure seamless integration and functionality within EBANA [25].

The adaptation process involved two key modifications:

1) **DC Simulation**: During DC simulations, the memristor's behavior is emulated through a behavioral resistor model, whose current-voltage characteristics is given by $I = I_0 \cdot e^{-\frac{\text{gap}}{g_0}} \cdot \sinh\left(\frac{V}{V_0}\right)$, where $I_0$, $g_0$, and $V_0$ are fitting parameters.

[1] https://keras.io

[2] At the time of this writing, the open-source Verilog-A compiler, OpenVAF, supports only compact models.

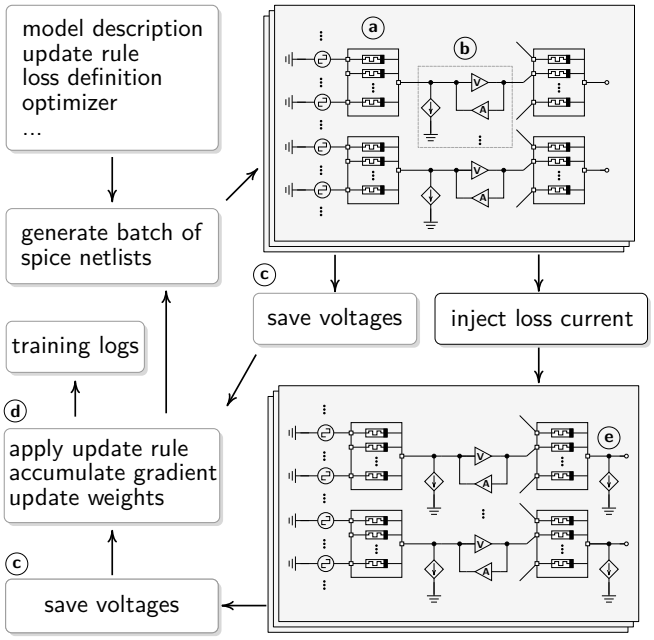



Fig. 3: EBANA training loop. The network's architecture, training parameters, and model files are first defined in Python. This information is then used to generate SPICE netlists for each training batch. Following the initial simulation, or free phase, the node voltages, and other relevant data, are extracted from the simulation log. The node voltages at the output nodes are compared with the true values using the selected loss function. The result is then translated to loss currents, which are injected into the output nodes. A second simulation, the nudge phase, is then performed, and the node voltages are measured again. Using the data from the two simulation phases, the weight gradients are calculated and accumulated. At the end of a batch, the accumulated gradients are used to update the weights. Note: Blocks marked with circled letters in this figure correspond with those in Fig. 2.

2) **Transient Simulation and State Update**: This process involves running a transient simulation on the memristor in Python. The final state is then used to update the DC model.

The memristor model offers a comprehensive set of fitting parameters, allowing the model to fit the shape of real memristor current-voltage (I-V) curves and resistance changes over time. A summary of some of the main parameters and the values used in our experiments is given in Table II.

The I-V characteristics of memristor defined in Table II is shown in Fig. 2f. Part (II) illustrates the characteristic butterfly curve, which represents the memristor's transition between low and high resistance states. This memristor exhibits a resistance range of approximately 100, aligning with what is typically observed in real memristors. The next two figures show how the resistance of the memristor evolves over time in response to a voltage pulse. The gray traces in these figures highlight the intrinsic cycle-to-cycle variation during programming.

TABLE II: Memristor parameteers.

| Param. | Value | Description |
|---|---|---|
| `I0` | $200 \cdot 10^{-9}$ | Affects the vertical positioning of the I-V curve. |
| `g0` | $0.32 \cdot 10^{-9}$ | Affects the resistance dynamic range. |
| `Vel0` | 5 | Affects the velocity at which the gap changes. |
| `Ea` | 0.6 | Affects gap change through the energy barrier. |

## IV. Results

In this section, we present our findings on critical aspects, including batching effects, different EP learning rules, gradient quantization, and the adaptation of update rules to memristor characteristics.

### A. Requirements For Batching

Neural networks are commonly optimized using gradient descent methods, which iteratively minimize the loss function, $C$, by updating the parameters, $\boldsymbol{\theta}$, in the direction of the steepest decline, as given in eq. 1. This update is performed by averaging the gradients across a mini-batch of $m$ samples and scaling by the learning rate $\alpha$.

$$\theta_k \leftarrow \theta_k - \frac{\alpha}{m} \sum_{n=1}^{m} \frac{\mathrm{d}C}{\mathrm{d}\theta_k}[n] \tag{1}$$

The batch size can significantly influence the training dynamics. Investigating the application of batching in the EP framework, particularly in analog computing, introduces complexities not fully explored in existing literature, prompting an examination of how batching advantages translate into this context.

The results from simulations on the Iris and MNIST datasets are presented in Figures 5a and 5b. The experiments reveal that larger batch sizes typically result in higher accuracies, supporting the notion that they better approximate the true gradient [26], even in the context of EP, where the calculated gradient for each sample is not exact as a result of approximating the EP update rule with a two point estimate [27].

In the Iris dataset, lower learning rates for a batch size of 1 enabled the network to achieve an accuracy of 95%, nearly matching the 96% accuracy observed for all batch sizes larger than 3. This demonstrates that with a sufficiently small learning rate, the network can take smaller steps towards a minima of a loss function. However, doing so required 1680 updates to be made, as opposed to 345 for the small-sized batch and 105 to the large-sized batch, for the best performing configurations. Conversely, for the MNIST dataset, even with significantly more updates (5000 compared to 100), the optimal configuration using a batch size of 1 only reached an accuracy of 87%, which is 7% lower than the accuracy achieved with a batch size of 40 after 5 epochs. This discrepancy between the MNIST and Iris datasets may be attributed to the inherent complexity of the tasks: The MNIST dataset represents a more complex classification task, necessitating

larger batch sizes to sufficiently capture the variability in the gradient.

### B. Requirements For Gradient Calculation

Within the Equilibrium Propagation (EP) framework, the weight parameters, $\boldsymbol{\theta}$, are updated exactly according to eq. 2, or approximated using a two-point derivative method, such as the forward difference approximation, to estimate the derivative for small values of $\beta$.

$$\Delta\boldsymbol{\theta} \propto \frac{\mathrm{d}}{\mathrm{d}\beta}\left(\frac{\partial F}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y}, \beta, \boldsymbol{s})\right) \tag{2}$$

Empirical results have shown that the bias introduced by the forward difference method leads to suboptimal performance in complex datasets, such as CIFAR-10 [27]. Various strategies have been proposed to mitigate this bias, such as changing the sign of $\beta$ randomly between updates [27], replacing the forward difference with the backward difference [28], and employing complex integration [29]. The first two methods require the computation of the circuit at only two values of $\beta$, whereas the third method, while can theoretically yield exact gradients, requires the computation of the circuit for multiple values of $\beta$, potentially imposing excessive energy demands. Consequently, the focus of this section is on the approximation of the EP update rule that requires only two-point computation.

The selection of the energy function $F$ required in the EP framework can be hand-crafted or derived from physical principles. For instance, in the context of two-terminal analog neural networks, if the energy function is chosen to be the sum of the pseudo-power of each component, as defined in eq. 3 [30], the minimum of this energy function can be shown to correspond to exactly the equilibrium state of the circuit [31].

$$F = \sum \int_0^{\Delta V} i(v) \cdot \mathrm{d}v \tag{3}$$

With this definition of the energy function, and using the forward difference approximation of eq. 2, it can be easily shown that the update equation of a linear resistor, with a current-voltage characteristic $i(v) = G \cdot v$, where $G$ is the conductance, is given by eq. 4. While accurately computing the difference between the squares of two signals, as eq. 4 suggests, is feasible, further exploration of simplified or alternative update rules could ease hardware implementation. We propose three additional update rules, summarized below, each with unique hardware implications, to be motivated and discussed in Sec. V.

$$\texttt{ep\_sq}: \Delta\theta \propto \lim_{\beta\to 0} \frac{1}{\beta}[(\Delta V^{[n]})^2 - (\Delta V^{[f]})^2] \tag{4}$$

$$\texttt{ep\_ana}: \Delta\theta \propto \lim_{\beta\to 0} \frac{1}{\beta}[(\Delta V^{[n]})^n - (\Delta V^{[f]})^n] \tag{5}$$

$$\texttt{ep\_abs}: \Delta\theta \propto \lim_{\beta\to 0} \frac{1}{\beta}[|\Delta V^{[n]}| - |\Delta V^{[f]}|] \tag{6}$$

$$\texttt{ep\_mult}: \Delta\theta \propto \lim_{\beta\to 0} \frac{2}{\beta}\Delta V^{[f]}(\Delta V^{[n]} - \Delta V^{[f]}) \tag{7}$$

Simulation results for the Iris and MNIST datasets are shown in Figures 5c and 5d. For the Iris dataset, all learning rules, except for `ep_abs`, demonstrated similar performance. On average, `ep_sq` and `ep_mult` performed the best (96% for both) and recorded the fastest convergence times. Interestingly, batch sizes 15 and 21 (5 and 7) were the best performers for `ep_sq` (`ep_mult`). This outcome is consistent with theoretical predictions, given that `ep_sq` is derived from the pseudo-power of a linear resistor. Contrary to expectations, on the MNIST dataset, `ep_ana` surpassed both `ep_sq` and `ep_mult` in performance, achieving a peak accuracy of 93% and a higher average ($\sim$ 92%) on the top 3 performing configurations. In terms of speed, all three rules achieved their best accuracies after 4 epochs.

The `ep_abs` learning rule has the worst performance on both datasets (average of $\sim$ 83% on Iris and $\sim$ 86% on MNIST on the top 3 performing configurations), taking almost twice as long to converge on Iris. This performance issue can be attributed to `ep_abs`'s tendency to overestimate the gradient for small voltages (below 0.5V), as explained in Fig. 4, resulting in larger weight updates. Consequently, when the model is close to the optimal solution, where precise, small updates are crucial, `ep_abs` applies excessive adjustments, causing the model to overshoot the minima. This is in contrast to the behavior exhibited by the other two rules, which underestimate the gradient for smaller voltages.

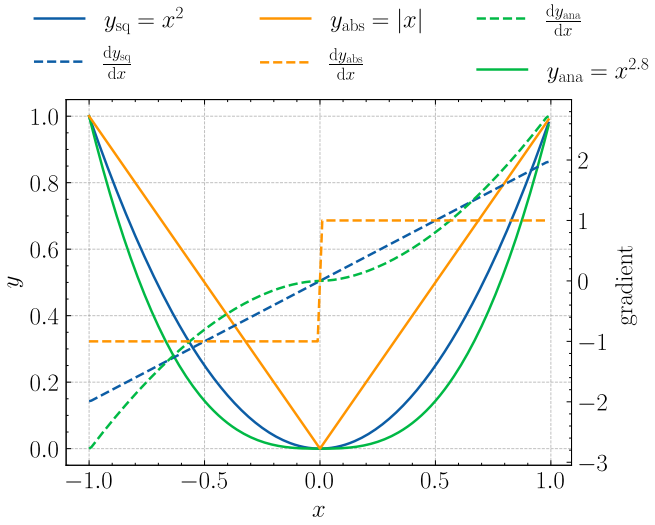



Fig. 4: Computation of the gradient. The gradient is computed in three steps. In the first two steps, we calculate $h^{[f]} = f(\Delta V^{[f]})$ and $h^{[n]} = f(\Delta V^{[n]})$, where $f$ corresponds to $(\cdot)^2$, $|\cdot|$, or $(\cdot)^{2.8}$, depending on the update rule used. Finally, we subtract the two values to get the gradient. Since the gradient is the difference between $h^{[n]}$ and $h^{[f]}$, we need to look at the derivative of the three rules to understand why `ep_ana` outperforms `ep_abs`. In the case of `ep_abs`, the gradient is overestimated for small voltages (below 0.5V) and underestimated for large voltages (above 0.5V). In contrast, `ep_ana` exhibits the opposite trend, which is advantageous, as it ensures smaller updates near the optimal point, thereby minimizing the risk of overshooting.

### C. Requirements For Gradient Quantization

The application of quantization in neural networks primarily falls into two categories: quantization for inference [32], and quantization for training. The interest of this work is in the latter. Unlike traditional approaches that compute gradients at one precision and perform updates at another [33], [34], our approach consistently applies 2-bit quantization, as described in eq. 8, reducing hardware complexity by eliminating dual precision pathways. This enables robust training on various datasets using mixed-signal processing with minimal fine-tuning.

$$\Delta\theta \propto \text{Quantize}(\Delta V^{[f]}, m) \cdot \Delta \tag{8}$$

This method contrasts with scenarios where high-precision multiplication precedes quantization or where multiplication involves low-precision numbers. Here, eq. 8 quantizes one multiplicand before multiplication, potentially explaining why the network converges with only 2-bits of precision. Although not practical for digital hardware, analog/mixed-signal hardware can accommodate this approach.

Implementing eq. 8 requires careful consideration of quantization techniques, range, and variability. Using 2-bit static uniform quantization on $\Delta V^{[f]}$ with a symmetric clipping range yields satisfactory results on the datasets we tested.

The simulations conducted on the Iris and MNIST datasets, illustrated in Figures 5e and 5f, demonstrate a predictable decrease in accuracy as the number of quantization bits in the calculation of the gradient decreases. Interestingly, some configurations with just 2 and 4 quantization bits achieve accuracies comparable to the unquantized best cases (98% for Iris and 96% for MNIST). Specifically, the top 3 performers for Iris have an average accuracy of 96.3% (95.6%) with batch sizes 21, 7, and 5, (105, 35, 105), 4 (2) bits of quantization, and requiring 25 (63) epochs on average. For MNIST, we observe a similar trend: both scenarios achieve a top accuracy of 92% with a batch size of 40, and 90% with a batch size of 10. Interestingly, one of the configurations with a 1 bit quantization level reached 91%, requiring 1 more epoch compared to the other two cases. This is 3% more than the top performing configuration for `ep_abs`. While this rule can also be regarding as a form of quantization, it has its quantization range pinned at $\pm 1$, making it less effective at quantizing the data.

In summary, these outcomes suggest two key insights: first, learning is feasible with gradient quantization using as few as 2 bits, potentially reducing hardware demands; and second, batching appears to enhance quantization efficiency, further supporting the argument for its implementation in hardware.

### D. Requirements For Memristor Update

While memristors are ideally expected to function as linear resistors, the reality of training circuits that incorporate memristors necessitates a careful consideration of their current-voltage (I-V) characteristics. Overlooking these characteristics can result in the inaccurate calculation of gradients, potentially hindering the network's ability to converge.

$$\Delta\theta \propto \lim_{\beta \to 0} \frac{1}{\beta} \left[ \frac{1}{2!} [(\Delta V^{[n]})^2 - (\Delta V^{[f]})^2] \right] \tag{9}$$

The results from simulations when using the exact equation of the memristor, as given in eq. 9 (derived in Sec. A-A), on the Iris and MNIST datasets are presented in Figures 5g and 5h. For the Iris dataset, the learning rules `ep_memristor` and `ep_analog` yielded the highest average accuracies. For the MNIST dataset, `ep_memristor` emerged as the top performer, with `ep_sq` and `ep_mult` a close second and third, on average. Interestingly, the top performer of `ep_mult`, an approximation of `ep_sq`, surpassed the top performer of `ep_sq` (94% vs 92%). These results show that that while learning is also feasible with other learning rules, optimizing the learning rule to align with the memristor's properties is crucial for achieving peak performance.

## V. Discussion

In this section, we discuss the hardware implications of our results and compare our method to the current state-of-the-art analog-based solutions.

### A. Hardware Implications of Implementing Batching

Although batching evidently enhances accuracy per update, the decision to implement it in hardware involves a deeper analysis beyond just the accuracy per update metric. Determining the appropriateness of batching in hardware requires consideration of four key factors:

1) The energy costs associated with batching.
2) The hardware costs incurred by batching.
3) The potential of batching to mitigate the imperfections inherent in analog computation.
4) The impact of batching on memristor longevity.

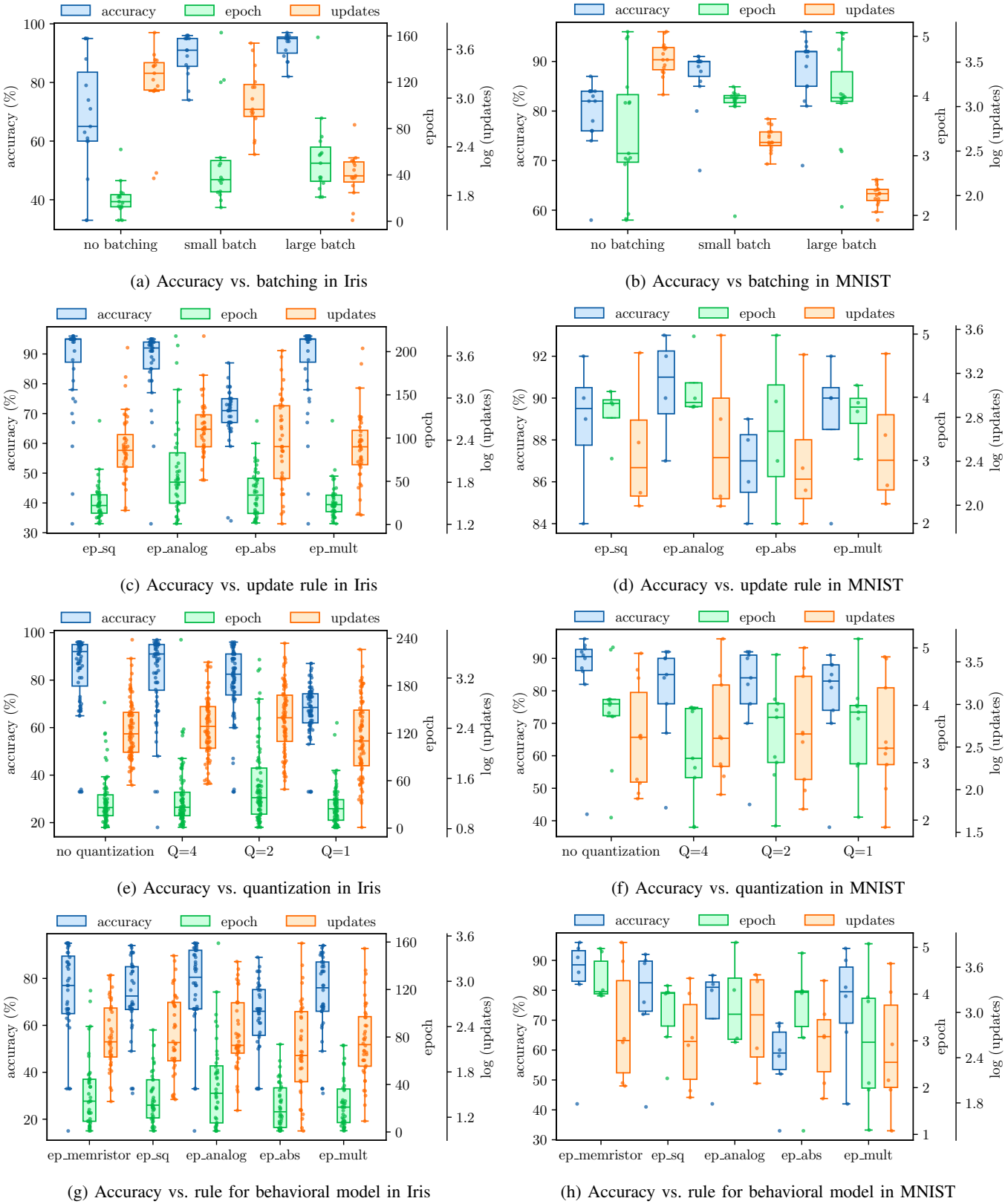
To evaluate the total energy consumption of a neural network employing batching versus one that does not, it is essential to account for two main operations: the energy associated with the accumulation of gradients for each sample within a batch and the energy required to update the weights at the end of a batch. The relevant parameters include:

- $n_d$: dataset size.
- $n_b$: batch size.
- $n_u$: number of updates to achieve the desired accuracy.
- $e_g$: energy required for accumulating the gradient per sample.
- $e_w$: energy required for updating the weights.

The system's total energy, $E_{\text{total}}$, is calculated as the sum of the energy for gradient accumulation, $E_{\text{grad}}$, and the energy for weight updates, $E_{\text{write}}$:

$$\begin{aligned} E_{\text{total}} &= \left( \frac{n_d}{n_b} \times n_u \right) \times n_b \times e_g + n_u \times e_w \\ &= \underbrace{n_d \times n_u \times e_g}_{E_{\text{grad}}} + \underbrace{n_u \times e_w}_{E_{\text{write}}} \end{aligned} \tag{10}$$

From an energy perspective, batching becomes advantageous when the total energy consumption with batching is




(a) Accuracy vs. batching in Iris

(b) Accuracy vs batching in MNIST

(c) Accuracy vs. update rule in Iris

(d) Accuracy vs. update rule in MNIST

(e) Accuracy vs. quantization in Iris

(f) Accuracy vs. quantization in MNIST

(g) Accuracy vs. rule for behavioral model in Iris

(h) Accuracy vs. rule for behavioral model in MNIST

Fig. 5: Aggregated results from the Iris and MNIST datasets showing the effects of batching (a, b), update equations (c, d), gradient quantization (e, f), and behavioral memristor training (g, h). Batching improves accuracy consistency and reduces updates needed. Update rules reveal that most rules, except `ep_abs`, achieve high accuracy. Gradient quantization with 2-bit levels require slightly longer training times but stable accuracy. Behavioral memristor training shows that `ep_memristor` achieves the highest accuracy, emphasizing the importance of aligning learning rules with memristor properties.

lower than that of non-batched training, expressed as $n_d \times n_{u1} \times e_g + n_{u1} \times e_w < n_{u2} \times e_w$, where $n_{u1}$ and $n_{u2}$ denote the number of updates with and without batching, respectively. Substituting $e_g = \gamma \cdot e_w$ and $n_{u1} = \zeta \cdot n_{u2}$, this condition simplifies to $\zeta(n_d \times \gamma + 1) < 1$, which further reduces to $\zeta \cdot \gamma < 1/n_d$. For example, comparing two configurations of the Iris dataset with batch sizes of 1 and 21, requiring 1259 and 80 updates respectively, batching is more energy-efficient if the gradient accumulation energy is less than $\sim 0.15$ times the weight update energy, calculated as $1/(\frac{80}{1259} \times 105) \approx 0.15$. This criterion provides a quantitative threshold for determining when the energy savings from batching justify its implementation.

Implementing batching in systems that use the EP method introduces additional hardware considerations. EP's algorithm requires measuring voltage drops across two training phases and storing these readings for gradient calculation, a requirement that persists with or without batching. From a hardware perspective, the additional cost in adding batching to a system that already implements the EP rule might hinge only on the additional requirements imposed by accumulating gradients across multiple samples. For instance, if a switched-capacitor accumulator is used as a component in the implementation of the EP rule, the additional cost for batching might be ensuring that the accumulator has enough output dynamic range to accommodate the aggregated gradient from multiple samples. Therefore, in such scenarios, the marginal hardware costs associated with batching may be considered reasonable and justified.

The potential of batching to mitigate the imperfections inherent in analog computation is another important consideration, particularly in the context where memristors are used for weight storage. Accurate gradient estimation is crucial in complex tasks, where precision in weight updates directly influences the network's performance and ability to converge [27]. Memristors, however, present challenges due to their non-linear and asymmetric response to programming pulses. By aggregating gradients from multiple training examples before updating the weights, batching not only averages out individual gradient errors but also reduces the frequency of weight adjustments. This aggregation process helps in compensating for the imperfect programming of memristors, thereby stabilizing the training dynamics. As a result, the risk of oscillatory behaviors, which can prevent convergence by continuously overshooting the optimal weights, is considerably reduced.

Finally, batching contributes to the longevity of memristor hardware. Memristors degrade with each programming operation; therefore, reducing the number of write operations through larger batch sizes directly extends the durability of the hardware, a key requirement for machine learning applications.

### *B. How To Implement The Different Update Rules*

We now discuss the implications of implementing each of the rules in hardware. The `ep_sq` and `ep_ana` rules will be examined together since they are related, followed by a separate discussion on the `ep_abs` rule.

The exact update rule according to the EP framework for a linear resistor is given by eq. 4, which calculates the difference of the square of two voltages, $\Delta V^{[f]}$ and $\Delta V^{[n]}$. These voltges represent the potential difference, $V_1 - V_2$, between two nodes of the circuit in the free and nudging phases, respectively.

Recall that the drain current, $I_D$, of an nMOS transistor operating in the saturation region is given by:

$$I_D = \frac{1}{2} \cdot K_n \cdot \left(\frac{W}{L}\right) \cdot (V_{\mathrm{GS}} - V_{\mathrm{TH}})^2, \tag{11}$$

where $K_n$ is a technology dependent parameter, $W/L$ is the aspect ratio of the transistor, $V_{\mathrm{GS}}$ is the gate-source voltage, and $V_{\mathrm{TH}}$ is the threshold voltage. If the two inputs $V_1$ and $V_2$ are applied to the gate and the source of a MOS transistor, the drain current is proportional to the square of the difference of the two inputs. Thus, the computation of the term $(\Delta V^{[f]})^2$ or $(\Delta V^{[n]})^2$ can be easily realized from the inherent square law of the MOS transistors operating in the saturation region.

In modern CMOS devices, factors such as velocity saturation, mobility degradation, and channel length modulation significantly alter the behavior of MOS transistors from the ideal square law. Thus, it becomes necessary to adapt the EP learning rule to align with the actual device characteristics, prompting the exploration of `ep_ana`. As an example, employing TSMC's 65nm process technology, we conducted a voltage sweep on the gate-source from $0.3$V to $1.0$V of a low-threshold nMOS transistor for transistors with lengths of 65nm and $2\mu$m from which we derived the corresponding drain currents. By fitting these data points to the model $I_D = k \cdot V_{\mathrm{GS}}^n$, we obtained values of $n \approx 2.9$ for the 65nm device and $n \approx 2.76$ for the $2\mu$m device. As a result, our simulations for `ep_ana` were done using $n = 2.8$ for a more realistic realisation of `ep_sq` using modern transistors.

Although our simulations indicate that `ep_ana` yields comparable performance to `ep_sq`, and while numerous circuits capable of executing the squaring operation exist [35], applying these circuits to realize the EP update rule presents significant obstacles.

One of the major challenges in using circuits that rely on the inherent square law characteristics of MOS transistors to calculate the square of the difference of two voltages is the problem of matching. For the circuits to operate over a wide range of voltages, these circuits usually employ two transistors, one for each voltage polarity. These are M1/M2 and M3/M4 in the circuit in Fig. 2g(I). As explained in [35], the current through the resistor $R_1$ is proportional to $(V_1 - V_2)^2$ provided that M1 and M2 are perfectly matched, requiring them to have the same threshold voltage at all times. Since the difference between $V_1$ and $V_2$ (nudged and free phase voltages) is usually very small, if the two transistors are not perfectly matched, the difference in the threshold voltages will be larger than the difference between $V_1$ and $V_2$, significantly impacting the calculation of the gradient. Moreover, such circuits must accurately square signals across both positive and negative values to avoid introducing biases that could skew gradient calculations. This requirement places stringent demands on the circuit's linearity across a wide voltage range, which is hard to achieve in analog.

The update rule `ep_abs`, which computes the absolute value of the voltage, offers a potential solution to the aforementioned problems while also being straight-forward to implement. A high level block diagram is shown in Fig. 2g(II). Only three simple blocks are required: a Swapping Circuit that swaps the position of $V_1$ and $V_2$, a comparator that checks for the condition $V_1 > V_2$, and a Charging Circuit to transfer the voltage to a storage element, such as a capacitor. Based on the combination of the signal $\varphi$ (which identifies the phase the circuit is in) and the output of the comparator, a simple logic inside the Swapping Circuit decides whether to swap the positions of $V_1$ and $V_2$. These signals go into the Charging Circuit, which could be a simple operational amplifier that transfers a charge proportional of the difference of the voltages onto the capacitor.

Although the implementation is straight-forward, simulation results show that this rule is inferior to the other three rules across both datasets. While it might be possible to find a sufficiently small learning rate that allows the system to converge, the amount of time, and consequently, the energy required might not be justified to opt for this solution.

To implement the gradient quantization rule with batching, we use the following expression, which follows from the `ep_mult` rule (derived in Sec. A-B):

$$V_{C_f} = \sum(V^{[f]}, \text{2-bit}, V_{\text{ref},p}, V_{\text{ref},m}) \cdot (V^{[n]} - V^{[f]}) \quad (12)$$

Before this equation is implemented in hardware, a few considerations need to be addressed, such as selecting between uniform and non-uniform quantization, defining the quantization range, and choosing between static and dynamic quantization. These considerations are essential for balancing accuracy and hardware complexity.

- Uniform vs. non-uniform quantization: Typically, non-uniform quantization offers greater accuracy for data that does not exhibit a uniform distribution. On hardware, this process requires an ADC that translates a sampled input into a digital code. With a known voltage distribution, implementing non-uniform quantization is straightforward; for example, by modifying the ladder resistance in low-bit resolution ADCs, such Flash ADCs.
- Selecting the quantization range: Quantization involves dividing an interval between $\alpha$ and $\beta$ into $2^m - 1$ parts. Various strategies exist for selecting $\alpha$ and $\beta$, each with its advantages and drawbacks regarding accuracy and implementation complexity. A simple approach sets $\alpha$ and $\beta$ at equidistant points from the origin. While this method is simple and ensures a tight clipping range, it may not always yield the best results. For instance, in EP, if negative voltages are not generated at the output of each layer, a method that can be used to reduce hardware complexity, the distribution of voltages skews towards the positive real axis the farther you move from the input layer, as demonstrated in Fig. 6. The optimal solution requires optimizing $\alpha$ and $\beta$ for each layer.
- Static or dynamic quantization: The variability in training inputs leads to diverse voltage distributions, making a single set of $\alpha$ and $\beta$ suboptimal across different inputs and different epochs. While computing the quantization ranges for each input often achieves higher accuracy, it can be very expensive to implement.

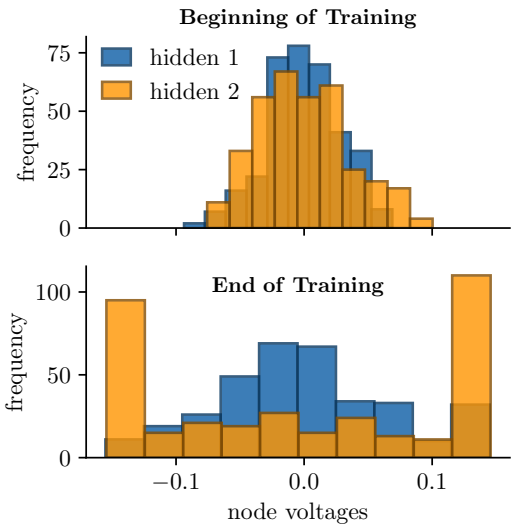



Fig. 6: Voltage distribution across nodes during the training of the Iris dataset with a three-layer neural network (two hidden layers and one output layer). Initially, node voltages are randomly distributed, mirroring the randomness in initial weights, with the second hidden layer showing a skew towards higher voltages. By the end of training, the distribution and scale of the voltages across the layers have significantly changed and stabilized, reflecting the network's learning progress.

Assuming that the optimal quantization parameters have been determined, eq. 12 can be implemented using an mDAC by rewriting it as follows:

$$v_{C_f} = \sum \Phi_\beta(\beta_1, \beta_2, S_1) \cdot \Phi_v(v^{[n]} - v^{[f]}, S_2) \quad (13)$$

Here, $\Phi_\beta(\beta_1, \beta_2, S_1)$ returns either $\beta_1$ or $\beta_2$ based on the state of the signal $S_1$, and $\Phi_v(v^{[n]} - v_{\text{ref}}, S_2)$ returns either $v^{[n]} - v^{[f]}$ or $-(v^{[n]} - v^{[f]})$ based on the state of the signal $S_2$.

A mixed-signal implementation of eq. 13 is illustrated in Fig. 2g(III) for the 2-bit case. It could be easily extended to support more bits. Its operation is explained below.

The process starts by acquiring the voltage drop, $V_{\text{mem}}$, across the memristor. During the free phase, when $\varphi_{\text{phase}}$ is LOW, $V_{\text{mem}}$ is passed to the 2-bit ADC, which outputs two control signals:

- $\varphi_{\text{sign}}$, which is HIGH for codes 10 and 11, indicating a positive input.
- $\varphi_{C2}$, which is HIGH for codes 00 and 11, and is used to enable capacitor $C_2$ in the mDAC.

In the mDAC, the voltage of the feedback capacitor, $V_{C_f}$, is calculated as:

$$V_{C_f} = \left(\frac{C_1 + \varphi_{C2} \cdot C_2}{C_f}\right) \cdot \Delta, \quad (14)$$

where $\Delta$ is the voltage stored in capacitors $C_1$ and $C_2$. If $C_1 = \alpha_1 \cdot C_f$ and $C_2 = \alpha_2 \cdot C_f$, the scaling of $\Delta$ depends on $\alpha_1$, $\alpha_2$ and the state of $\varphi_{C2}$. When $\varphi_{C2}$ is HIGH, $\Delta$ is scaled by $\beta_1 = (\alpha_1 + \alpha_2)$. When $\varphi_{C2}$ is LOW, $\Delta$ is scaled by $\beta_2 = \alpha_1$.

The sign of $\Delta$ is adjusted based on the state of the switches, $S_1$ and $S_2$, in the switched-capacitor circuit, which operates in two phases: charging and accumulation, controlled by $\varphi_{\text{sc}}$. The operation is as follows:

During the free phase ($\varphi_{\text{phase}}$ is HIGH), $V_{\text{mem}}$ is equal to $V^{[f]}$.

- In the charging phase ($\varphi_{\text{sc}}$ is LOW), the bottom plates of $C_1$ and $C_2$ are connected to $V_{\text{mem}}$, and the top plates are grounded, storing a voltage of $-V^{[f]}$ in the capacitors.
- In the accumulation phase ($\varphi_{\text{sc}}$ is HIGH), the bottom plates are grounded, and the top plates are connected to the inverting node of the amplifier, transferring a voltage of $-\beta_1 \cdot V^{[f]}$ or $-\beta_2 \cdot V^{[f]}$, depending on the state of $\varphi_{C2}$.

During the nudging phase ($\varphi_{\text{phase}}$ is HIGH), $V_{\text{mem}}$ is equal to $V^{[n]}$.

- In the charging phase, the top plates of $C_1$ and $C_2$ are connected to $V_{\text{mem}}$, and the bottom plates are grounded, storing $V^{[n]}$.
- In the transfer phase, the bottom plates are grounded, and the top plates are connected to the inverting node of the amplifier, transferring a voltage of $\beta_1 \cdot V^{[n]}$ or $\beta_2 \cdot V^{[n]}$.

The above description allows for the implementation of $\beta_1(V^{[n]} - V^{[f]})$ or $\beta_2(V^{[n]} - V^{[f]})$ based on the sign of $\varphi_{C2}$. For the other possibilities, $-\beta_1(V^{[n]} - V^{[f]})$ or $-\beta_2(V^{[n]} - V^{[f]})$, we apply $V^{[f]}$ to the top plates and $V^{[n]}$ to the bottom plates. This decision is made by Switch Logic circuit based on the output of the ADC, the training phase (free or nudge), and the phase of the switched-capacitor circuit (charging or transfer).

In conclusion, by adopting 2-bit precision for gradient quantization across all stages, our simulations demonstrate that it is possible to achieve robust training performance across the two tested datasets. This result offers a promising avenue for implementing EP on resource-constrained devices. Finally, by adopting a mixed-signal approach, gradient quantization and batching can be integrated together.

### C. Hardware Implications For the Memristor Update Circuit

The method used to update a memristor varies depending on the type of the memristor and the design of the array architecture [36]. Resistive RAM (RRAM) is particularly favored in crossbar arrays for neural computations due to its ability to achieve distinct resistance states through the growth and dissolution of conductive filaments within its active layer. This process can be achieved through various approaches. For example, architectures that use a transistor as a selector device apply the programming voltage to the transistor's gate to set the current during the SET operation. Alternatively, modulation of the programming voltage for a fixed duration, modulation of the pulse duration at a fixed voltage, or a combination of these methods are used [37]. Consequently, the voltage driving circuits must be designed to provide adjustable voltage range and variable pulse widths to accommodate different programming schemes.

The implementation of RRAM-based memristors as storage elements comes with challenges, primarily due to the inherent variability in the formation and dissolution of the conductive filaments, as shown in Figure 2. These challenges are typically addressed through iterative write-verify programming schemes or weight slicing methods [38], [39], which not only consume significant power and silicon area but are also time-consuming. Despite these issues, the precise control over conductance levels is less critical for certain applications. In contexts like ours, where the system supports online training, inaccuracies in the memristor's writing process can be corrected dynamically, effectively mitigating issues that are common in conventional digital memory systems or neural networks using memristive crossbars in a digital configuration.

In view of this, we investigate a very simple update rule, called the Manhattan Rule, that uses only the sign of the gradient [40]. The Manhattan method is very easy to implement in hardware as there is no need to convert the weight update values to precise writing voltages, a very difficult process thanks to the nonlinearly-changing conductance of the memristor. The most basic form of this rule simply produces a negative (positive) pulse of fixed duration when the gradient is positive (negative). Since this always results in either a positive or a negative pulse, the accuracy can bounce back and forth as the network gets closer to convergence. This issue can be fixed by only producing a pulse when the magnitude of the gradient exceeds a certain threshold. Such an update rule could be implemented with only two comparators. If the threshold voltage is fixed, the hardware requirements could be further simplified by integrating the threshold voltages directly in the comparators, a technique commonly-used to design biased comparators. This leads to more compact hardware implementation and faster training time. This is the method we used to update the memristors.

### D. Comparative Analysis

In an effort to optimize the equilibrium propagation algorithm for memristor-based hardware, extensive experiments were conducted to identify the best parameter configurations. Our aim in this section is to demonstrate that our selected parameters not only match the accuracy of an idealized software implementation but also surpass the performance of existing hardware solutions, without the degradation observed in prior implementations.

Our approach is informed by a practical architecture with real hardware constraints and supported by rigorous empirical analysis on two datasets. For the Iris dataset, employing 4-bit gradient quantization achieved 95% accuracy, vastly superior to the 63% achieved by the method of [22]. Similarly, for the MNIST dataset, our 4-bit quantization approach reached 93% accuracy, compared to only 48% with their method. Please note that attempts to replicate the results of [22] in our experiments failed to yield comparable outcomes, highlighting potential issues with the robustness and general applicability of their approach.

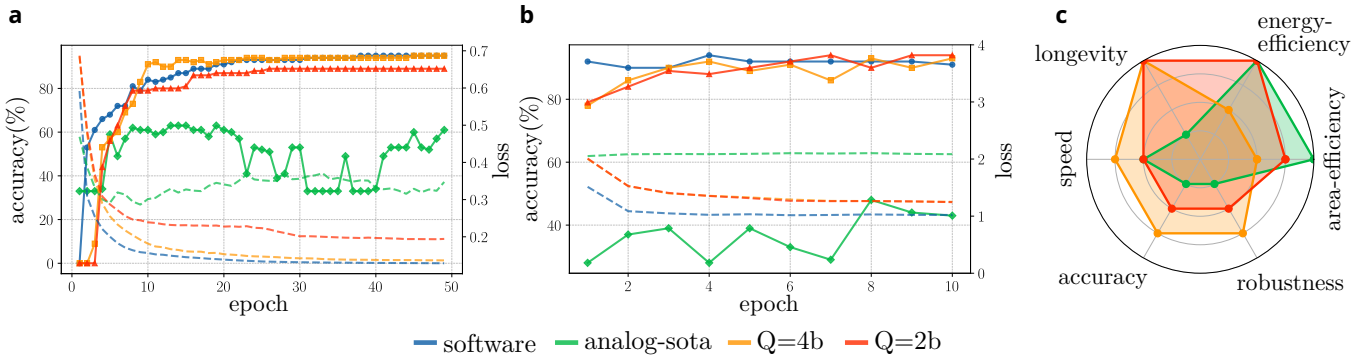



Fig. 7: Comparison Plot. The plot labelled "software" corresponds to the scenario where the simulation is conducted in software. The plot labelled "analog-sota" corresponds to the method suggested by [22]. The final two plots demonstrate our proposed method, using 4-bit and 2-bit gradient quantization with batching enabled. For all experiments, we selected only those configurations (best learning rate, best batch size, etc.,) that yielded the best performance. The "software" experiment was conducted using a behavioral model, the `ep_memristor` update rule, and exact memristor programming. The other experiments were conducted using the memristor model, their respective update rules, and the simplified memristor programming approach. **(a)**. Iris Training: Using 4-bit gradient quantization yielded an accuracy of 95%. With 2-bit quantization, accuracy dropped to 89%. The method proposed by [22] achieved a maximum accuracy of 63%, but the learning process was unstable. **(b)**. MNIST Training: Using 4-bit gradient quantization yielded an accuracy of 93%. With 2-bit quantization, accuracy dropped to 94%. The method proposed by [22] achieved a maximum accuracy of 48%, but the learning process was unstable. **(c)**. Qualitative plot across various performance metrics.

For a qualitative analysis of other performance metrics, we assessed the area efficiency, energy efficiency, memristor longevity, and robustness. Due to the lack of quantitative data from the authors of [22], we adopted a qualitative scoring system ranging from poor (1) to excellent (4), as shown in Fig. 7. In terms of area efficiency, our design required more area due to the use of capacitors and additional transistors for implementing the 4-bit ADC, whereas the design by [22] used no capacitors, suggesting a minimal area requirement. Energy efficiency comparisons showed that our 4-bit approach had higher energy demands due to the need for 15 comparators in the FLASH ADC structure, whereas the 2-bit and the design by [22] had comparable energy requirements. Memristor longevity was significantly better in our design due to less frequent updates facilitated by batching, contrary to the design by [22] that updated memristors with every sample. Robustness was also enhanced in our method, which consistently performed well across different configurations, unlike the design by [22] which showed greater susceptibility to instability.

## VI. Conclusion

This study advances the implementation of Equilibrium Propagation (EP) within analog machine learning, addressing key challenges in hardware deployment, a topic often overlooked. Our SPICE-level experiments on two datasets using different resistor models reveal several key insights. While techniques such as no batching and a simplified EP update rule using the absolute function showed individual promise, their combination with standard memristor update rules resulted in instability across both datasets. However, using a combination of 2-bit gradient quantization and effective batching, we achieved an accuracy of 93% on the MNIST dataset and 95% on the Iris dataset, outperforming the state-of-art hardware implementable approach, which only achieved 48% accuracy on MNIST and 63% on Iris.

Moreover, we propose an architecture template and gradient calculation circuits for end-to-end hardware realization. Our flexible architecture, which leverages mixed-signal processing and resource-sharing techniques, optimizes computational performance and power efficiency. The proposed hardware solution inherently performs the batching operation, improving training accuracy and extending hardware longevity.

Despite these promising results, our experiments were limited to two datasets and a simple neural network architecture. Furthermore, challenges such as noise effects, signal degradation, and voltage refresh circuits remain unaddressed, which are critical for EP's successful deployment in real-world applications. Future research should take these into account and include a broader array of datasets and more complex network configurations to confirm the scalability of our solutions. Looking ahead, refining EP integration within memristive crossbar arrays to maximize energy efficiency and computational throughput will be a key focus. Additionally, exploring adaptive learning rates to mimic advanced optimization algorithms like ADAM, without the associated hardware overhead, offers a promising direction to improve network convergence and validate EP's practical viability.

## Appendix A
## Derivations

### A. Memristor Update Equation Derivation

We have shown in Sec. IV-B that a circuit implementing EP should ideally be able to calculate the gradient according eq. 15. However, while this is the ideal update equation for

a linear resistor, and while under specific cases can lead to convergence, it is not the optimal update equation for a memristor.

$$\Delta\theta \propto \lim_{\beta\to 0} \frac{1}{\beta}[(\Delta V^{[n]})^2 - (\Delta V^{[f]})^2] \tag{15}$$

We now derive the precise update rule for the particular memristor model we're using. The DC I-V characteristic equation of the memristor is given by:

$$I = I_0 \cdot e^{-\frac{\text{gap}}{g_0}} \cdot \sinh\left(\frac{V}{V_0}\right) \tag{16}$$

$$= \underbrace{\frac{I_0 \cdot e^{-\frac{\text{gap}}{g_0}}}{V_0}}_{G} \cdot \sinh\left(\frac{V}{V_0}\right) \cdot V_0 \tag{17}$$

We can use `gap` as the programming variable and derive the exact update rule for eq. 16. This will give a gradient of the form $\frac{\mathrm{d}\mathcal{P}}{\mathrm{dgap}}$. However, to compare the update rule for the memristor with the update rule given in eq. 15, we rewrite eq. 16 as shown in eq. 17 so that $G$ (for conductance) is the programming variable. Applying eq. 3 and the forward-difference formula approximation of eq. 2 results in:

$$\Delta\theta \propto \lim_{\beta\to 0} \frac{V_0^2}{\beta}\left[\cosh\left(\frac{\Delta V^{[n]}}{V_0}\right) - \cosh\left(\frac{\Delta V^{[f]}}{V_0}\right)\right] \tag{18}$$

Using the Taylor series of $\cosh(x)$, eq. 18 can be rewritten as:

$$\Delta\theta \propto \lim_{\beta\to 0} \frac{1}{\beta}\left[\frac{1}{2!}[(\Delta V^{[n]})^2 - (\Delta V^{[f]})^2] + \frac{1}{4!\cdot V_0^2}[(\Delta V^{[n]})^4 - (\Delta V^{[f]})^4] + \cdots\right] \tag{19}$$

Disregarding terms with exponents greater than 4, the update rules for a linear resistor and memristor with characteristics given by eq. 17 differ by a term equal to $\frac{2}{4!\cdot V_0^2}[(\Delta V_\beta)^4 - (\Delta V_0)^4]$, which becomes less significant with smaller voltages. To make the second term in eq. 19 be $\gamma$ times the first term, the voltages must remain below $\pm\sqrt{12\gamma V_0^2}$. For example, to reduce the second term to 10% of the first, the voltage limit is approximately $\pm 0.27$V. This could enable the application of eq. 16 over eq. 19 in the update rule, thereby greatly simplifying hardware requirements. However, for a voltage of approximately $\pm 0.87$V, the gradient calculated by eq. 16 is nearly half of the optimal value.

### *B. `ep_mult` Rule Derivation*

The results of the simulations based on rule `ep_mult` follow very closely those from rule `ep_sq`. This is not surprising since `ep_mult` is a specific approximation of rule `ep_sq`. It can be deduced as follows:

$$\Delta\theta \propto (\Delta V^{[n]})^2 - (\Delta V^{[f]})^2$$

Let $\Delta = \Delta V^{[n]} - \Delta V^{[f]}$. Therefore:

$$\Delta\theta \propto 2 \cdot \Delta V^{[f]} \cdot \Delta - \Delta^2$$

If $\Delta$ is very small, $\Delta^2 \to 0$, leading to the update rule:

$$\Delta\theta \propto 2 \cdot \Delta V^{[f]} \cdot \Delta \tag{20}$$

The novelty of this rule is that it not only aligns closely with the theoretical optimum but also circumvents the practical limitations of implementing `ep_sq`. Instead of requiring a circuit that computes the square of the difference of two numbers, we require a circuit that multiplies two numbers. With some further approximations, as will be discussed in the next section, this rule can be easily implemented using a multiplicative capacitive DAC (shown in Fig. 2c).

## REFERENCES


[1] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamara, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: concepts, cnn architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, no. 1, Mar. 2021. [Online]. Available: https://doi.org/10.1186/s40537-021-00444-8

[2] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, jul 2017. [Online]. Available: https://doi.org/10.1109%2Fmsp.2017.2693418

[3] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 3645–3650. [Online]. Available: https://aclanthology.org/P19-1355

[4] N. N. Alajlan and D. M. Ibrahim, "TinyML: Enabling of inference deep learning models on ultra-low-power IoT edge devices for AI applications," *Micromachines*, vol. 13, no. 6, p. 851, may 2022. [Online]. Available: https://doi.org/10.3390%2Fmi13060851

[5] S. Zhu, T. Voigt, J. Ko, and F. Rahimian, "On-device training: A first overview on existing systems," 2023.

[6] L. Li, D. Shi, R. Hou, H. Li, M. Pan, and Z. Han, "To talk or to work: Flexible communication compression for energy efficient federated learning over heterogeneous mobile edge devices," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. IEEE, may 2021. [Online]. Available: https://doi.org/10.1109%2Finfocom42981.2021.9488839

[7] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, "Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 3, pp. 881–892, sep 2021. [Online]. Available: https://doi.org/10.1109%2Ftccn.2021.3066619

[8] M. Tsukada, M. Kondo, and H. Matsutani, "A neural network-based on-device learning anomaly detector for edge devices," *IEEE Transactions on Computers*, pp. 1–1, 2020. [Online]. Available: https://doi.org/10.1109%2Ftc.2020.2973631

[9] X. Liu and Z. Zeng, "Memristor crossbar architectures for implementing deep neural networks," *Complex & Intelligent Systems*, vol. 8, no. 2, pp. 787–802, Apr. 2022.

[10] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, "Efficient processing of deep neural networks: A tutorial and survey," 2017. [Online]. Available: https://arxiv.org/abs/1703.09039

[11] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, jun 2016. [Online]. Available: https://doi.org/10.1109%2Fisca.2016.13

[12] S. Zahrai and M. Onabajo, "Review of analog-to-digital conversion characteristics and design considerations for the creation of power-efficient hybrid data converters," *Journal of Low Power Electronics and Applications*, vol. 8, no. 2, p. 12, apr 2018. [Online]. Available: https://doi.org/10.3390%2Fjlpea8020012

[13] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, jun 2016. [Online]. Available: https://doi.org/10.1145%2F2897937.2898010

[14] T. Chou, W. Tang, J. Botimer, and Z. Zhang, "CASCADE," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, oct 2019. [Online]. Available: https://doi.org/10.1145%2F3352460.3358328

[15] B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, and H. Yang, "RRAM-based analog approximate computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 12, pp. 1905–1917, dec 2015. [Online]. Available: https://doi.org/10.1109%2Ftcad.2015.2445741

[16] B. Scellier and Y. Bengio, "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation," *Frontiers in Computational Neuroscience*, vol. 11, p. 24, May 2017.

[17] J. C. R. Whittington and R. Bogacz, "An Approximation of the Error Backpropagation Algorithm in a Predictive Coding Network with Local Hebbian Synaptic Plasticity," *Neural Computation*, vol. 29, no. 5, pp. 1229–1262, 05 2017. [Online]. Available: https://doi.org/10.1162/NECO_a_00949

[18] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random feedback weights support learning in deep neural networks," 2014.

[19] G. Hinton, "The forward-forward algorithm: Some preliminary investigations," 2022.

[20] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, "Backpropagation and the brain," *Nature Reviews Neuroscience*, vol. 21, no. 6, p. 335–346, Apr. 2020. [Online]. Available: http://dx.doi.org/10.1038/s41583-020-0277-3

[21] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *CoRR*, vol. abs/1806.07366, 2018. [Online]. Available: http://arxiv.org/abs/1806.07366

[22] S. Oh, J. An, S. Cho, R. Yoon, and K.-S. Min, "Memristor Crossbar Circuits Implementing Equilibrium Propagation for On-Device Learning," *Micromachines*, vol. 14, no. 7, p. 1367, Jul. 2023.

[23] M. Watfa, A. Garcia-Ortiz, and G. Sassatelli, "Energy-based analog neural network framework," *Frontiers in Computational Neuroscience*, vol. 17, Mar. 2023. [Online]. Available: http://dx.doi.org/10.3389/fncom.2023.1114651

[24] S. University, "Stanford university resistive-switching random access memory (rram) verilog-a model 1.0.0," https://nanohub.org/publications/19/1, 2014.

[25] M. Watfa, "Python memristor models," https://github.com/medwatt/python_memristor_models, 2024.

[26] D. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural Networks*, vol. 16, no. 10, pp. 1429–1451, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608003001382

[27] A. Laborieux, M. Ernoult, B. Scellier, Y. Bengio, J. Grollier, and D. Querlioz, "Scaling equilibrium propagation to deep convnets by drastically reducing its gradient estimator bias," 2020.

[28] B. Scellier, M. Ernoult, J. Kendall, and S. Kumar, "Energy-based learning algorithms for analog computing: a comparative study," 2023.

[29] A. Laborieux and F. Zenke, "Holomorphic equilibrium propagation computes exact gradients through finite size oscillations," 2022.

[30] J. Kendall, R. Pantone, K. Manickavasagam, Y. Bengio, and B. Scellier, "Training end-to-end analog neural networks with equilibrium propagation," *arXiv:2006.01981 [cs]*, Jun. 2020.

[31] W. Johnson, "Nonlinear electrical network," https://sites.math.washington.edu/~reu/papers/2017/willjohnson/directed-networks.pdf.

[32] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," Jun. 2021.

[33] L. Cambier, A. Bhiwandiwalla, T. Gong, M. Nekuii, O. H. Elibol, and H. Tang, "Shifted and squeezed 8-bit floating point format for low-precision training of deep neural networks," 2020.

[34] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 5151–5159.

[35] H.-J. Song and C.-K. Kim, "An mos four-quadrant analog multiplier using simple two-input squaring circuits with source followers," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 3, pp. 841–848, 1990.

[36] A. Prakash and H. Hwang, "Multilevel cell storage and resistance variability in resistive random access memory," *Physical Sciences Reviews*, vol. 1, no. 6, Jun. 2016.

[37] S. Yu, Y. Wu, and H.-S. P. Wong, "Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory," *Applied Physics Letters*, vol. 98, no. 10, p. 103514, Mar. 2011.

[38] S.-S. Sheu, M.-F. Chang, K.-F. Lin, C.-W. Wu, Y.-S. Chen, P.-F. Chiu, C.-C. Kuo, Y.-S. Yang, P.-C. Chiang, W.-P. Lin, C.-H. Lin, H.-Y. Lee, P.-Y. Gu, S.-M. Wang, F. T. Chen, K.-L. Su, C.-H. Lien, K.-H. Cheng, H.-T. Wu, T.-K. Ku, M.-J. Kao, and M.-J. Tsai, "A 4mb embedded slc resistive-ram macro with 7.2ns read-write random-access time and 160ns mlc-access capability," in *2011 IEEE International Solid-State Circuits Conference*, 2011, pp. 200–202.

[39] W. Song, M. Rao, Y. Li, C. Li, Y. Zhuo, F. Cai, M. Wu, W. Yin, Z. Li, Q. Wei, S. Lee, H. Zhu, L. Gong, M. Barnell, Q. Wu, P. A. Beerel, M. S.-W. Chen, N. Ge, M. Hu, Q. Xia, and J. J. Yang, "Programming memristor arrays with arbitrarily high precision for analog computing," *Science*, vol. 383, no. 6685, pp. 903–910, Feb. 2024.

[40] E. Zamanidoost, F. M. Bayat, D. Strukov, and I. Kataeva, "Manhattan rule training for memristive crossbar circuit pattern classifiers," in *2015 IEEE 9th International Symposium on Intelligent Signal Processing (WISP) Proceedings*, 2015, pp. 1–6.