Efficient Implementation of Spiking Neural Networks using Memristors

Ishan Jha*
IIIT-Bangalore, India
Email: Ishan.Jha@iiitb.ac.in

Aryan Mishra*
IIIT-Bangalore, India
Email: aryan.mishra@iiitb.ac.in

Ananda Y R
IIIT-Bangalore, India
Email: ananda.yr@iiitb.ac.in

***Abstract*—Spiking Neural Networks (SNNs) are a class of artificial neural networks inspired by the way biological neurons process and transmit information. Unlike traditional artificial neural networks (ANNs), which use continuous values for activations, SNNs use discrete events called spikes to encode and process information over time. These spikes are similar to the electrical impulses observed in biological neurons. SNNs seem to outperform artificial neural networks (ANNs) on the basis of multiple parameters, such as energy efficiency, temporal dynamics, temporal information processing and biological plausibility.**

Drawing inspiration from the architecture and operating principles of the human brain, spiking neural networks (SNNs) represent the next evolution of artificial neural models. They have garnered widespread interest thanks to their ultra-low energy, pulse-based signaling, and exceptional capacity for massive parallel processing. Although neural network research is increasingly shifting from software simulations toward dedicated hardware realizations, this transition poses substantial hurdles. Among potential hardware platforms, memristors stand out for their rapid programming, minimal power requirements, and seamless integration with CMOS processes. In this review, we first outline the fundamental workings of SNNs, then explore how memristor-based approaches can enable their hardware deployment, and examine how customized algorithmic optimizations might enhance both efficiency and energy economy in SNN hardware. We conclude by surveying current challenges and open questions surrounding the use of memristor technology in this domain.

***Index Terms*—Spiking Neural Networks (SNNs), Artificial Neural Networks (ANNs), Memristors, Neuron**

I. Introduction

Comprising billions of neurons linked by trillions of synapses, the human brain forms an extraordinarily intricate and highly interconnected network, making it arguably the most powerful biological processor of information on the planet. In this sophisticated architecture, neurons function both as detectors—receiving inputs from the environment or other cells, and as processors, performing the computations that underlie perception and cognition. Synapses, in turn, act as specialized junctions that transmit precisely coded patterns of electrical and chemical signals between neurons, enabling the seamless flow of information throughout the system. It is this elegant interplay of structure and function that grants the brain its remarkable abilities: from rigorous logical reasoning and deep abstract thinking to spontaneous bursts of creativity. Perhaps most astonishingly, all of these complex parallel processing tasks are performed with astonishing energy efficiency, as the entire organ operates on only about 20 watts, comparable to a dim household light bulb, demonstrating an unparalleled balance of computational power and power conservation. [7]

Over the past several years, research and development in the domain of artificial neural networks, which are computational models inspired by the interconnected neurons and synapse architecture of the human brain, have accelerated at an unprecedented pace. Advances in training algorithms, the availability of massive datasets and the exponential growth of computational power have collectively enabled these systems to tackle increasingly complex tasks, from natural language understanding and image recognition to strategic decision-making in dynamic environments. As a result, modern neural architectures have not only become deeper and more efficient but have also given rise to entirely new sub-fields, such as convolutional and transformer based networks, driving transformative applications across industries ranging from healthcare diagnostics to autonomous vehicles. In contrast to the continuous signal processing utilized in conventional artificial neural networks (ANNs), spiking neural networks (SNNs) operate using discrete pulse-like signals to transmit information. Rather than responding to every individual spike as it occurs, neurons within an SNN integrate the effects of multiple incoming pulses over time. Only when the accumulated input reaches a certain threshold does the neuron fire and transmit a spike to connected neurons, a process referred to as event-triggered communication. This biologically inspired mechanism significantly reduces redundant activity and allows SNNs to function with remarkable efficiency. As a result, spiking neural networks exhibit notable advantages in terms of computational efficiency and energy conservation. These benefits have been consistently demonstrated across a wide range of applications, including neuromorphic computing, real-time processing in embedded systems, and low-power edge devices.

Despite the inherent advantages of spiking neural networks (SNNs) in terms of energy efficiency, their potential has not been fully realized due to constraints imposed by the traditional von Neumann computing architecture. This architecture, which separates memory and processing units, introduces a bottleneck in data transfer and significantly limits the energy-saving benefits that SNNs are capable of offering. As a result, the full promise of SNNs in achieving ultra-low-power com-

* The authors contributed equally to this research work.

putation remains largely untapped in conventional hardware platforms. In contrast, a new wave of emerging technologies that support in-memory computing paradigms where data storage and processing occur within the same physical location have demonstrated much greater potential for efficient SNN implementation. Technologies such as phase-change memory (PCM), resistive random access memory (RRAM), magnetic random access memory (MRAM), and ferroelectric memory provide more biologically plausible and energy-efficient substrates for neuromorphic computing. These non-von Neumann memory technologies not only reduce latency and energy overhead associated with data movement but also align more naturally with the spike-based, event-driven nature of SNNs, paving the way for the development of high-performance, low-power neuromorphic hardware systems. Among them, RRAM (that is, memristors) has become a strong candidate to stimulate the key units (neurons and synapses) of SNNs. [7]

One of the most promising and intriguing features of using two-terminal memristor arrays as synaptic elements in neural networks lies in their ability to overcome fundamental limitations of the traditional von Neumann computing architecture. In conventional systems, computation and data storage are physically separated into distinct units, processors, and memory, which necessitates constant and energy-intensive data transfer between the two. This separation not only introduces significant latency but also leads to increased power consumption, especially in the case of neural networks that rely heavily on memory access for high volume, parallel processing. The bandwidth bottleneck becomes a critical challenge when scaling such networks for real-time or energy-sensitive applications.

In contrast, the human brain demonstrates a remarkably efficient model of information processing in which memory and computation coexist within the same physical regions, that is, within the synapses and neurons. Inspired by this biological efficiency, a new paradigm known as "in-memory computing" has emerged. This architectural shift allows for data processing and storage to occur simultaneously in the same location, effectively eliminating the need for frequent data shuttling between separate units.

Memristor arrays, particularly those based on two-terminal devices, are uniquely well-suited for implementing this paradigm. By leveraging fundamental physical principles such as Ohm's law and Kirchhoff's current law, these devices can perform parallel analog computations directly within the memory structure. When applied to the hardware implementation of spiking neural networks (SNNs), memristor-based synapses can facilitate real-time signal processing with drastically reduced latency and power consumption. This makes them a highly attractive solution for building scalable, energy-efficient neuromorphic systems that more closely mimic the processing dynamics of the human brain.

Although some review articles have already explored the application of memristors in the context of neural networks, focusing on areas such as the emulation of synaptic functions, the integration of memristors into network training and inference processes, and their deployment in various neuromorphic computing applications, certain critical challenges remain that must be addressed to fully realize their potential. Despite the considerable progress made in this field, memristors are still subject to several intrinsic non-idealities that hinder their performance and scalability in practical implementations.

Among the most prominent limitations is the restricted number of discrete conductance states that many memristor devices can reliably support. This constraint reduces the precision with which synaptic weights can be represented, affecting the accuracy and learning capacity of memristor-based neural networks. Additionally, memristors often exhibit nonlinear conductance modulation during programming, which complicates the implementation of precise weight updates, particularly in gradient-based learning algorithms. Another major concern is device-to-device and cycle-to-cycle variability, which introduces randomness and inconsistency into the system, making it difficult to achieve reproducible behavior across large-scale arrays.

These non-ideal characteristics pose significant challenges for designing robust and high-performance neuromorphic systems. As such, ongoing research efforts are increasingly focused not only on improving the materials and fabrication techniques of memristors to mitigate these drawbacks but also on developing new algorithms and circuit-level compensation strategies that can tolerate or even exploit such imperfections. Addressing these issues is essential for advancing the practical application of memristor-based hardware in large-scale, efficient, and biologically inspired neural computing platforms.

II. Spiking Neural Networks

According to Wolfgang Maass, neural networks can be broadly categorized into three evolutionary generations, each representing a significant advancement in computational modeling and biological plausibility. The first generation comprises simple architectures such as the perceptron, which perform binary threshold operations. These models output discrete values—typically zeros or ones—based on whether the weighted sum of inputs exceeds a predefined threshold. Although foundational, these early models were limited in their ability to represent complex, non-linear relationships.

To address these limitations, the second generation introduced continuous, non-linear activation functions, such as the widely known sigmoid and hyperbolic tangent (tanh) functions. These enabled neural networks to output a continuous range of values, greatly enhancing their expressive power. Crucially, this generation laid the groundwork for the development of modern deep neural networks (DNNs), as the introduction of differentiable functions allowed the use of gradient-based optimization techniques such as backpropagation (BP), which revolutionized the training of multilayer networks.

The third generation of neural networks marks a radical departure from the previous two, particularly in the way information is encoded and transmitted. Known as spiking neural networks (SNNs), these models replace continuous or discrete

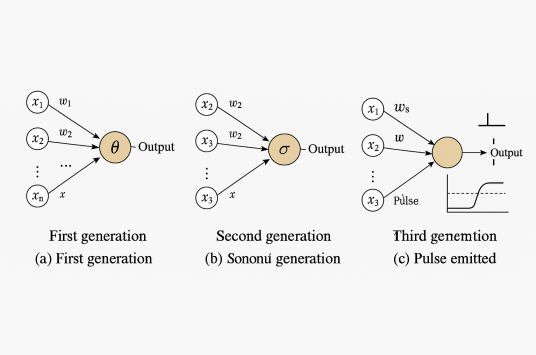

Fig. 1. Diagrammatic Comparison between the 3 generations of neural networks

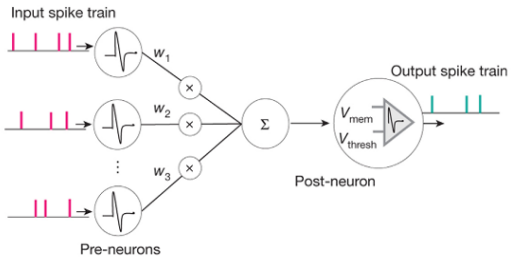

Fig. 2. Structure of a classical SNN

numerical signals with temporally sparse spike trains, which are discrete pulses that are more analogous to the behavior of biological neurons. Although the architectural components of SNNs, neurons, synapses, and network topologies, resemble those of traditional artificial neural networks (ANNs), their operational principles are fundamentally different. The diagrammatic representation showing the difference between the three generation of neural networks in shown in Fig.

In traditional ANNs and DNNs, neurons receive real-valued inputs from the previous layer, which are weighted, summed, and immediately passed through an activation function such as ReLU or sigmoid to produce the output. Each neuron typically participates in the computation at each time step, leading to a high computational load and energy usage.

However, SNNs adopt an event-driven paradigm. Here, neurons receive pulse-based inputs that are also weighted and summed, but instead of immediately applying an activation function, the neuron integrates the incoming signals over time. This accumulation continues until the internal state of the neuron reaches a certain threshold, at which point it emits a spike or pulse to the next layer. This temporal integration mimics the behavior of biological neurons and allows SNNs to operate asynchronously.

One of the key advantages of this design is that not all neurons and synapses need to be active during each computational cycle. Instead, only those neurons that have accumulated sufficient input will fire, drastically reducing the number of operations and, consequently, the energy consumption. This asynchronous and sparse activity makes SNNs particularly attractive for low-power real-time applications, such as those found in neuromorphic hardware and edge computing. As illustrated in Fig. , this biological inspiration allows SNNs to offer a more energy efficient alternative to conventional neural networks, while still supporting sophisticated computational capabilities.

A. Neurons

Numerous spiking neuron models have been developed to emulate the behavior of biological neurons, each offering varying degrees of biological fidelity and computational efficiency. These models exist on a continuum of biological mimicry, from highly detailed biophysical representations to abstracted mathematical approximations optimized for large-scale simulations. Among the most biophysically accurate is the Hodgkin–Huxley (HH) model, which rigorously characterizes the ionic mechanisms underlying the generation and propagation of action potentials in neurons. The HH model explicitly incorporates the passive electrical properties of the neuronal membrane and models the voltage- and time-dependent conductance changes associated with sodium (Na+) and potassium (K+) ion channels through a set of nonlinear differential equations. This formulation enables the model to reproduce a wide range of electrophysiological behaviors observed in empirical recordings, thereby serving as a gold standard for high-fidelity neuron modeling. [3]

Despite its precision, the HH model imposes a substantial computational burden due to its complex system of equations and the fine temporal resolution required for accurate numerical integration. This makes it infeasible for deployment in large-scale spiking neural networks (SNNs) or energy-constrained neuromorphic hardware. [2] To alleviate these limitations, more computationally efficient models have been introduced. One such model is the Izhikevich neuron model, which retains key dynamical features of biological neurons while significantly reducing computational cost. This model describes neuronal dynamics using a two-variable system: a membrane potential variable and a recovery variable. The recovery variable encapsulates the combined effects of slow ionic currents, particularly the delayed rectifier K+ current and the inactivation of Na+ channels, effectively implementing a biologically motivated feedback mechanism. Despite its simplified formulation, the Izhikevich model is capable of reproducing a wide array of spiking and bursting patterns characteristic of real neurons, including tonic firing, phasic spiking, and complex bursting behaviors, making it a compelling trade-off between biological plausibility and simulation efficiency. [4]

However, in many applied contexts, especially in large-scale neuromorphic systems or real-time embedded implementations, the level of biological mimicry afforded by models like HH or Izhikevich may be unnecessary and even detrimental due to their energy and computational demands. [1] Therefore, it becomes imperative to consider the application-specific balance between neurobiological realism and implementation cost. In this context, the Leaky Integrate-and-Fire (LIF) model has gained widespread adoption due to its minimalistic but

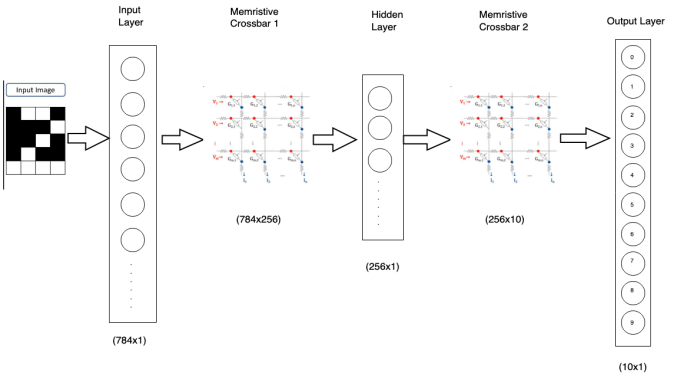

Fig. 3. Block diagram of the proposed CIM-SNN with adaptive threshold modulation.

functionally effective representation of neuronal behavior. The LIF model describes membrane potential dynamics using a first-order differential equation that integrates incoming synaptic currents over time while incorporating a leak term to account for passive decay. When the membrane potential reaches a defined threshold, a spike is emitted and the potential is reset to a resting state, mimicking the essential spike generation mechanism of real neurons. [5]

Due to its analytical tractability and low computational complexity, the LIF model is particularly well suited for implementation in large-scale SNNs and neuromorphic platforms. [8] It enables efficient simulation of network dynamics while maintaining the temporal precision necessary for spiking-based information processing. In addition, the LIF model supports a variety of extensions, including refractory periods, conductance-based synapses, and adaptation mechanisms, which further enhance its flexibility without significantly compromising efficiency. Its ability to strike a pragmatic balance between fidelity and resource consumption underpins its ubiquity in both theoretical studies and practical applications of spiking neural computation. [8]

The membrane potential can be described by the formulas given in Fig. and the meaning of the symbols used are given in Fig. [7].

III. Proposed Architecture

The proposed architecture augments the baseline memristor-based Computing-In-Memory(CIM) SNN [6] by introducing an adaptive threshold modulation mechanism in the ISI encoder, and specifying the full data-flow from input to classification. Figure 3 shows the high-level block diagram of our design.

A. Adaptive Threshold Modulation

Fixed, static firing thresholds are ill-suited to variable or noisy inputs: they either fail to fire on weak signals or produce spurious spikes under large noise excursions. To address this, we provide each input-layer LIF neuron with an *input-dependent* threshold plus a *dynamic* homeostatic mechanism, as shown in Figure 3.

1) Instantaneous Input-Dependent Threshold: At each simulation timestep, the firing threshold $V_{\mathrm{th}}(t)$ of every input-layer LIF neuron is updated based on its drive current:

$$V_{\mathrm{th}}(t) = V_{\mathrm{th,base}} + k\, I_{\mathrm{in}}(t)\,, \tag{1}$$

where

- $V_{\mathrm{th,base}}$ is the fixed baseline threshold (e.g. 1 mV),
- $I_{\mathrm{in}}(t)$ is the instantaneous input current, proportional to the (noisy) pixel intensity,
- k is the gain factor (e.g. 0.1 mV/pA), chosen to control how much the threshold shifts.

With this rule, large currents (whether from strong signals or noise) push the threshold upward, preventing unwanted spikes, while small currents lower the threshold, preserving sensitivity to faint inputs.

2) Dynamic Threshold Evolution: To prevent excessive rapid firing while preserving responsiveness, the firing threshold V_{th} is updated in two simple steps:

(a) **Spike-triggered increase:** Immediately after each output spike, the threshold is bumped up:

$$V_{\mathrm{th}} \;\leftarrow\; V_{\mathrm{th}} + \Delta V_{\mathrm{th}} \tag{2}$$

(b) **Exponential decay:** Between spikes, the threshold relaxes back toward its baseline $V_{\mathrm{th,base}}$:

$$\tau_{\mathrm{th}} \frac{\mathrm{d}V_{\mathrm{th}}}{\mathrm{d}t} = V_{\mathrm{th,base}} - V_{\mathrm{th}}(t) \tag{3}$$

Here,

- ΔV_{th} (e.g. 0.5 mV) is the per-spike threshold increment,
- τ_{th} (e.g. 30 ms) is the time constant controlling how fast the threshold decays back.

With this rule, each spike makes the neuron momentarily less excitable—avoiding bursts—and then shortly afterward the threshold returns to its normal value, readying the neuron for new inputs.

B. Input Layer: ISI Encoding with Adaptive Threshold

The input layer converts each normalized pixel intensity $x_i \in [0,1]$ into a pair of spikes whose inter-spike interval (ISI) carries the value of x_i. To do this:

(1) **Current mapping with noise.**

$$I_i(t) = I_{\max}\, x_i \;+\; \eta_i(t), \quad \eta_i(t) \sim \mathcal{N}(0, \sigma^2),$$

where we set $\sigma = 0.1\, I_{\max}$ (i.e. 10% Gaussian noise).

(2) **Parallel LIF integration.** Each noisy current drives two identical LIF neurons in parallel:

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -V + R\, I_i(t), \quad \text{fire if } V > V_{\mathrm{th}}(t),\; V \leftarrow V_{\mathrm{reset}}. \tag{4}$$

The threshold $V_{\mathrm{th}}(t)$ follows our adaptive rule in Eq. (III-A1).

(3) **Two-spike ISI generation.** Each neuron emits exactly two spikes per input cycle. We record

$$\Delta t_i \;=\; t_i^{(2)} \;-\; t_i^{(1)},$$

the time between the first spike $t_i^{(1)}$ and the second spike $t_i^{(2)}$.

(4) **ISI → digital code.** The interval Δt_i is monotonically related to x_i, giving a high-precision temporal code.

Because the threshold adapts to the same noisy current, this ISI remains stable even when η_i is large.

The resulting two-spike patterns (one pair per pixel) form the spike-interval input to the memristor crossbar in the next stage, combining high information density with strong noise robustness.

C. Memristor Crossbar (CIM) Layer

In this stage, we perform the vector–matrix multiplication directly inside the memory array using a memristor crossbar. The key ideas are:

1) **Voltage encoding:** Each incoming ISI-encoded spike pair is converted into an analog voltage V_i on the corresponding word-line.
2) **Conductance weights:** Each memristor at crosspoint (i, j) holds a conductance G_{ij} that encodes the trained synaptic weight.
3) **Kirchhoff's current summation:** According to Kirchhoff's current law, the total current flowing out of bit-line j is
$$I_j \;=\; \sum_i G_{ij}\, V_i. \tag{5}$$
4) **Massive parallelism:** All columns compute their dot products simultaneously, realizing an N-by-M multiply–accumulate in one step.

Because computation happens where the weights are stored, this CIM approach eliminates data movement overhead, yielding high throughput and low energy per operation [7].

D. Hidden Layer and ISI Re-encoding

The hidden layer repeats the same three-stage pipeline as the input layer, which makes it straightforward to add more layers:

1) **Current integration:** Each column output current I_j from the crossbar (Eq. (5)) is delivered to one of 256 LIF neurons, each with a fixed threshold.
2) **Two-spike generation:** Each LIF neuron integrates its input according to
$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -V + R\, I_j(t), \quad \text{fire if } V > V_{\mathrm{th}},\; V \leftarrow V_{\mathrm{reset}},$$
and emits exactly two spikes. The timing between these spikes carries the information of I_j.
3) **ISI re-encoding:** The interval Δt_j between the two spikes is measured by the same extractor circuit used in the input layer. This produces a new two-spike ISI code for each neuron.

The resulting ISI-encoded spike pairs serve directly as the input voltages for the next memristor crossbar stage. Because each hidden layer uses this identical "integrate $\rightarrow$ two-spike LIF $\rightarrow$ ISI extractor" sequence, layers can be stacked modularly to build deep networks.

E. Output Layer: TTFS Decoder

The output layer converts the final analog currents into a single-spike temporal code for fast, low-power classification:

1) **Current amplification.** Each bit-line current I_j from the last crossbar is passed through a simple current-mirror amplifier to ensure it is large enough to drive a spiking neuron.
2) **Single LIF integration.** The amplified current feeds a single LIF neuron:
$\tau_m \dfrac{\mathrm{d}V}{\mathrm{d}t} = -V + R\, I_j(t), \quad$ fire when $V > V_{\mathrm{th}},\; V \leftarrow V_{\mathrm{reset}}$.
Each neuron is allowed exactly one spike per input cycle.
3) **Time-to-first-spike (TTFS) readout.** We record the moment $t_j^{(1)}$ when neuron j emits its first (and only) spike.
4) **Label selection.** The network's predicted class is the index j^* whose neuron spiked earliest:
$$j^* = \arg\min_j\, t_j^{(1)}.$$

Because only one spike is generated per neuron, this TTFS scheme minimizes both inference latency and energy consumption, making it ideal for real-time, low-power applications.

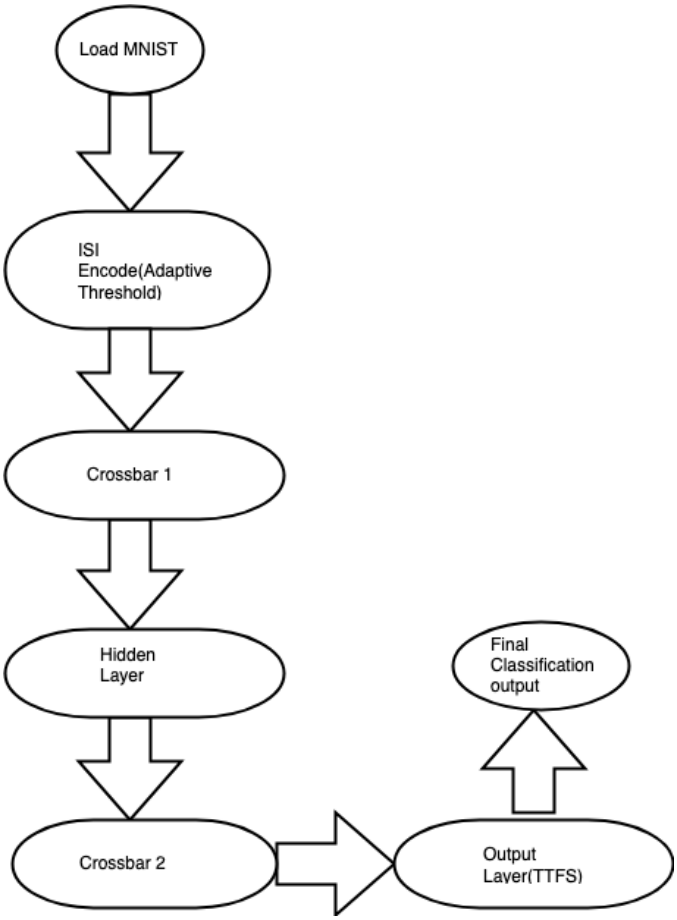

Fig. 4. End-to-end data-flow of the adaptive-threshold CIM-SNN.

F. Complete Workflow

The complete working and the execution flow is shown in Fig 4.

IV. Theoretical Analysis

1) Noise Robustness: By making the threshold depend on the same input noise, the effective noise term in the spike-

generation decision becomes

$$\eta_{\text{net}} = \eta + k\,\eta = (1+k)\,\eta,$$

so its variance is

$$\text{Var}(\eta_{\text{net}}) = (1+k)^2\,\text{Var}(\eta).$$

Therefore, the ratio of the original noise variance to the new noise variance is

$$\frac{\text{Var}(\eta)}{\text{Var}(\eta_{\text{net}})} = \frac{1}{(1+k)^2}, \tag{6}$$

which is strictly less than 1 for any $k > 0$. For instance, with $k = 0.1$,

$$\frac{1}{(1+0.1)^2} \approx 0.82,$$

i.e. a $\sim 18\%$ reduction in effective noise variance.

A. Power Overhead

Adding adaptive threshold logic incurs an estimated overhead of 5% in input-unit power.

V. Implementation and Experimental Setup

A. Simulation Environment

We implemented the proposed adaptive-threshold CIM-SNN entirely in Python (v3.8) using the following libraries:

- **NumPy** for general numerical operations.
- **PyTorch** for tensor manipulation and neural-network modules.
- **SnnTorch** to provide surrogate-gradient support for spiking neurons.
- **torch.utils.data** for batching and dataset management.
- **Keras MNIST loader** to fetch and preprocess the MNIST dataset.
- **Brian2** for detailed neuron-and-network simulation.

All simulations used a fixed timestep of 1 µs to capture precise spike timings and threshold dynamics.

Key parameters are summarized in Table I.

TABLE I
KEY SIMULATION PARAMETERS

Parameter	Value
Membrane time constant, τ_m	15 ms
Membrane resistance, R	1 MΩ
Baseline threshold, $V_{\text{th,base}}$	1.5 mV
Threshold gain, k	0.1 mV/pA
Crossbar conductance range	10^{-6}–10^{-3} S
Noise standard deviation, σ	0.01 $I_{\max}$
Simulation timestep	1 µs

B. Dataset and Noise Model

We benchmark our model on the standard MNIST dataset of handwritten digits, which consists of 60 000 training and 10 000 test images of size 28×28 with pixel values in $[0, 1]$. Prior to training, all images are normalized to the unit interval.

a) Training: The network weights are learned on the *clean* MNIST training set using the Adam optimizer (learning rate 10^{-3}, batch size 128) for 10 epochs. After training, these weights are frozen for subsequent noise-robustness evaluation.

b) Noise Injection: To assess performance under degraded input, we generate a *noisy* test set by adding zero-mean Gaussian noise to each pixel:

$$x_i' = \text{clip}\big(x_i + \eta_i\big), \quad \eta_i \sim \mathcal{N}\big(0,\,(0.1)^2\big),$$

where clipping ensures $x_i' \in [0, 1]$. This corresponds to a noise standard deviation of 10% relative to the maximum pixel intensity.

c) Evaluation: We compare the classification accuracy of fixed-threshold vs. adaptive-threshold ISI encoders on the noisy test set, highlighting the benefit of adaptive threshold modulation under input variability.

VI. Results and Discussion

We evaluate the proposed adaptive-threshold CIM-SNN on two key metrics: classification accuracy under noisy inputs, and overall power consumption.

A. Classification Accuracy

On the MNIST test set with 10% additive Gaussian noise, the baseline fixed-threshold ISI encoder achieves 78% accuracy. By applying the adaptive threshold modulation from Section III, accuracy rises to 88%, an absolute gain of 10 percentage points (a 12.8% relative improvement). This demonstrates that dynamically adjusting each neuron's threshold to its instantaneous input effectively suppresses noise-induced spikes and preserves sensitivity to weak signals.

B. Power Consumption

We estimate total system power by summing contributions from all 784 input neurons, crossbar operations, hidden-layer neurons, and output decoders. The fixed-threshold design consumes 48.22 mW, while the adaptive-threshold variant consumes 50.63 mW—an increase of just 2.41 mW (approximately 5%). This modest overhead is due to the simple arithmetic and comparator circuitry required to compute $V_{\text{th}}(t)$ in Eq. (III-A1), and remains well within the constraints of low-power edge applications.

VII. Conclusion

We have presented a novel adaptive threshold modulation technique for ISI encoding in a memristor-based CIM SNN. By making each neuron's firing threshold an instantaneous function of its input current, our design achieves a 10% absolute improvement in noisy-MNIST accuracy, with only a 5% power overhead and no additional latency. Future work will focus on silicon prototyping of the threshold logic, integrating on-chip learning for dynamically tuned thresholds, and validating the approach on real-world CIM hardware platforms.

References

[1] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[2] W. Gerstner and W. M. Kistler, *Spiking neuron models*. Cambridge, England: Cambridge University Press, Jun. 2012.

[3] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *J. Physiol.*, vol. 117, no. 4, pp. 500–544, Aug. 1952.

[4] E. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003.
[5] W. Maass and C. M. Bishop, *Pulsed neural networks*. MIT press, 2001.
[6] F. Nowshin and Y. Yi, "Memristor-based deep spiking neural network with a computing-in-memory architecture," in *2022 23rd International Symposium on Quality Electronic Design (ISQED)*. IEEE, Apr. 2022.
[7] H. Peng, L. Gan, and X. Guo, "Memristor-based spiking neural networks: cooperative development of neural network architecture/algorithms and memristors," *Chip*, vol. 3, no. 2, p. 100093, Jun. 2024.
[8] B. Schrauwen, D. Verstraeten, and J. Campenhout, "An overview of reservoir computing: Theory, applications and implementations," 01 2007, pp. 471–482.