Catwalk: Unary Top-K for Efficient Ramp-No-Leak Neuron Design for Temporal Neural Networks

Devon Lister
University of Central Florida
Orlando, FL, USA
devon.lister@ucf.edu

Prabhu Vellaisamy
Carnegie Mellon University
Pittsburgh, PA, CMU
pvellais@andrew.cmu.edu

John Paul Shen
Carnegie Mellon University
Pittsburgh, PA, CMU
jpshen@andrew.cmu.edu

Di Wu
University of Central Florida
Orlando, FL, USA
di.wu@ucf.edu

***Abstract*—Temporal neural networks (TNNs) are neuromorphic neural networks that utilize bit-serial temporal coding. TNNs are composed of columns, which in turn employ neurons as their building blocks. Each neuron processes volleys of input spikes, modulated by associated synaptic weights, on its dendritic inputs. Recently proposed neuron implementation in CMOS employs a Spike Response Model (SRM) with a ramp-no-leak (RNL) response function and assumes all the inputs can carry spikes. However, in actual spike volleys, only a small subset of the dendritic inputs actually carry spikes in each compute cycle. This form of sparsity can be exploited to achieve better hardware efficiency. In this paper, we propose a Catwalk neuron implementation by relocating spikes in a spike volley as a sorted subset cluster via unary top-k. Such relocation can significantly reduce the cost of the subsequent parallel counter (PC) for accumulating the response functions from the spiking inputs. This can lead to improvements on area and power efficiency in RNL neuron implementation. Place-and-route results show Catwalk is $1.39\times$ and $1.86\times$ better in area and power, respectively, as compared to existing SRM0-RNL neurons.**

***Index Terms*—temporal neural network, neuron, temporal coding, unary computing, top-k, hardware efficiency.**

I. Introduction

Neuromorphic computing, particularly represented by spiking neural networks (SNNs) [15], has emerged as an attractive alternative to conventional compute-intensive deep neural networks (DNNs) due to its brain-inspired approach for computational efficiency. Unlike DNNs, which transmit continuous-valued signals, neurons in SNNs communicate information through discrete spike or pulse events. Central to these architectures is the neuron model, which defines how neurons generate and respond to spikes. Among various neuron models, the spike response model (SRM) has been widely utilized, especially in its simplified variant, the SRM0 model, characterized by a fixed spike generation threshold. The SRM0 neuron can employ several types of response functions, such as biexponential [4], piecewise linear [6], step-no-leak [3], [5], [12], and, notably, the ramp-no-leak (RNL) function [12], [13]. The RNL function has recently gained significant interest due to its practical advantages in temporal neural network (TNN) applications [1], [7], [8], [12]–[14], [17]. TNNs, a special class of SNNs, are capable of continuous [13] online learning and unsupervised clustering [1], [12] by employing precise spike timing and biologically-plausible spike-timing dependent plasticity (STDP) local learning rule. This contrasts with the compute intensive global backpropagation methods predominantly used in traditional DNNs.

979-8-3315-3477-6/25/$31.00 © 2025 IEEE

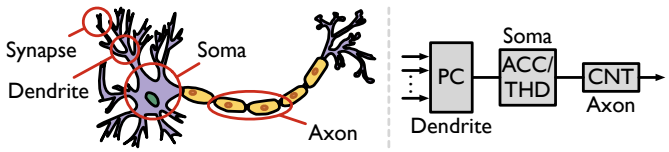
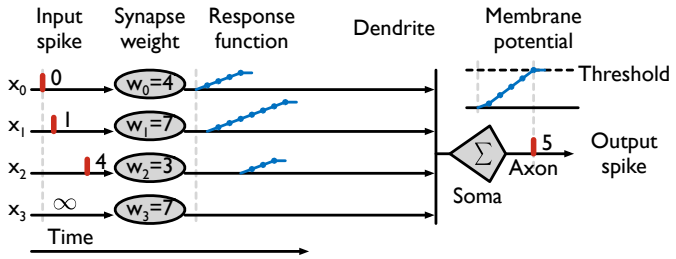


Fig. 1: A bilogical neuron and and its circuit representation using ramp-no-leak response function. PC (parallel counter) integrates all current incoming spikes modulated by the synaptic weights. ACC/THD is the soma (neuron body) that accumulates (ACC) all incoming potentials from dendrites and checks whether the accumulated potential surpasses a predefined threshold (THD). CNT (counter) is the axon output that fires a spike if the accumulated potential is higher than the threshold. The synapses are not shown here for simplicity.

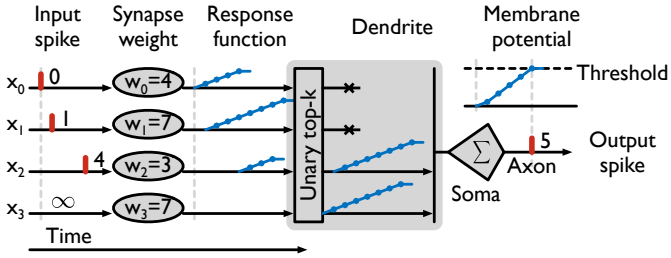
Existing TNNs adopt an SRM0 neuron model with an RNL response function [12], [13]. For brevity, we henceforth call this neuron model an *SRM0-RNL neuron* in this paper. However, current SRM0-RNL neuron implementations [7], [8], [17] employ worst-case scenarios for temporal signal processing and provides increased hardware resources for maximal spike density scenarios, which rarely occurs due to the inherent sparsity, i.e., just 0.1%-10% of total count of neurons are observed to fire biologically [10], [11], [20].

To address these inefficiencies, this paper introduces *Catwalk*, a novel neuron architecture employing unary top-k to optimize spike aggregation at neuron inputs. Catwalk leverages the inherent sparsity in temporal-coded spikes to cluster active spikes efficiently, enabling the replacement of conventional, fully provisioned parallel counter (PC), for accumulation of input response functions, with significantly more compact and efficient counterparts. Specifically, our contributions are:

- We propose Catwalk, a novel technique that relocates the top-k temporal spikes to optimize spike aggregation in neuron inputs for better hardware efficiency.
- A detailed evaluation against baseline designs, including SRM0-RNL neurons that employ conventional PCs and those that integrate unary sorting, demonstrates the clear



(a) An existing SRM0-RNL neuron model [12], [13]. The input spikes $x_{\{0,1,2\}}$ are temporal-coded (red pulses), where the spike timing encodes the value. Note it is allowed that an input has no spike at all, representing a value of ∞, e.g., $x_{\{3\}}$. When a synapse receives an input spike, it will instantly trigger the RNL response function, which is described in Equation 1. This function generates a pulse whose width is identical to the weight value, indicated by the number of dots in blue. The soma will accumulate all these weight pulses as the membrane potential. Once the accumulated potential exceeds a threshold, the axon will fire an output spike, and the neuron will be reset.



(b) Our proposed Catwalk neuron model with unary top-k. Catwalk inserts unary top-k at the dendrite, which relocates and clusters the spikes together. Our spike relocation reduces the number of valid spike volleys without alternating the number of valid spikes, reducing the cost of PC in the dendrite.

Fig. 2: Comparison of an existing SRM0-RNL neuron model and our proposed Catwalk neuron model for TNNs.

advantages of our proposed Catwalk neuron.

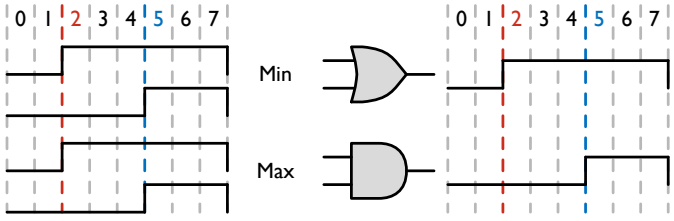
- Place-and-route evaluation at 45nm CMOS shows area and power improvements of $1.39\times$ and $1.86\times$, respectively, relative to existing SRM0-RNL neurons.

This paper is organized as follows. Section II and Section III review the background and motivate this work. Then Section IV describes the proposed neuron design. The following Section V and Section VI evaluate the implementation. Finally, Section VII concludes this work.

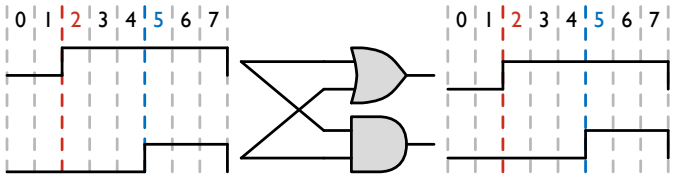
II. Background

A. SRM0-RNL Neuron

SRM0-RNL neurons closely follow the formulation of biological neurons (Fig. 1), and existing TNNs encode information through precise spike timings rather than spike rates [12], [13]. The mechanism of this neuron model is explained further in Fig. 2a from prior works [12], [13], and is contrasted with our Catwalk neuron model, depicted in Fig. 2b. Each post-synaptic neuron is fed by incoming temporal-coded spikes from all preceding neurons (pre-synaptic neurons), where the synapse responses are accumulated. The axon fires a spike if the accumulated membrane potential exceeds a certain threshold. In the existing SRM0-RNL implementations, all incoming



(a) Min and max via temporal coding.



(b) Compare-and-swap unit via min and max. This is also a 2-input bitonic sorter. Sorting more inputs can be done by repeating this unit recursively.

Fig. 3: Unary sorting via temporal coding. This example encodes unary data in a leading-0 mode. The timing of the rising edge marks the data value, as colored by red and blue.

responses are summed during potential accumulation [7]. The RNL response function is provided in Equation 1, where w is the weight value and t refers to time. Therefore, the RNL response function essentially creates a pulse of width w over time, to be accumulated later.

$$\begin{aligned} \rho(w,t) &= 0 && \text{if } t < 0 \\ &= t+1 && \text{if } 0 \le t < w \\ &= w && \text{if } t \ge w \end{aligned} \tag{1}$$

B. Unary Sorting

Unary data representations, either rate-coded or temporal-coded, have been used to perform various operations, including multiplication, addition [18], division, square root [19], min, max [16]. Among these, bitonic sorting on temporal-coded unary data (e.g. temporal spikes in SRM0-RNL neurons) can be implemented via simple AND and OR gates [9]. An example of unary sorting is shown in Fig. 3. Fig. 3a shows the use of simple AND and OR gates to perform min and max operations with temporal coding, respectively. Fig. 3b illustrates the basic compare-and-swap unit for unary sorting. The input temporal signals are ranked in order at the output, with the larger values clustered at the bottom. The simplicity of unary compare-and-swap units offers opportunities in integration with SRM0-RNL neurons, leading to a more optimized design.

III. Motivation and Opportunity

The existing SRM0-RNL neuron design assumes the worst-case scenarios. For an n-input neuron, the PC must accumulate all n inputs, even if temporal spikes are not present for some inputs. This worst-case neuron design does not consider the biological aspect of neurons, i.e. it is de facto that neuron spikes are extremely sparse and only $0.1\% \sim 10\%$ of total neurons are actively spiking in any given compute cycle [10],

[11], [20]. Such a worst-case neuron design overprovisions hardware resources, incurring suboptimal efficiency.

Since the inputs of SRM0-RNL neurons are temporal coded, there exist opportunities to find the inputs with effective spikes using the unary logic in Fig. 3, and then use a more lightweight PC to aggregate these effective spikes.

In this work, we propose Catwalk to take advantage of this existing technique and optimize spike aggregation for more efficient SRM0-RNL neurons in TNNs. Our proposed Catwalk neuron model ingests all temporal spikes from pre-synaptic neurons and relocates them, so that all active spikes are clustered together. Then we can substitute the original full PC with a more lightweight version. As long as the cost of the spike relocation and the new PC is less than the original full PC, Catwalk offers hardware efficiency gains. Given the high neuronal sparsity within actual workloads, Catwalk should not cause significant accuracy concerns. More experimental work is needed to validate this.

IV. Catwalk Neuron via Unary Top-K

In this section, we describe the hardware design of the proposed Catwalk neuron using unary top-k.

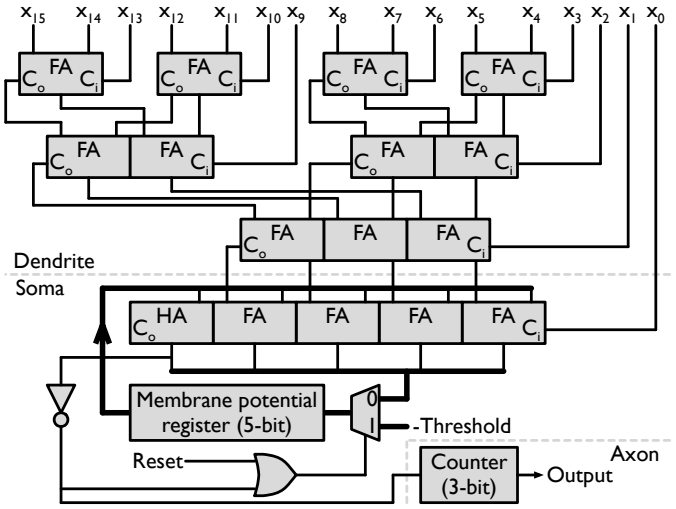
A. *Catwalk Neuron Microarchitecture*

We show the microarchitecture of the existing SRM0-RNL neuron and our Catwalk neuron with unary top k in Fig. 4. As illustrated in Fig. 2b, the key idea of Catwalk is to identify the valid temporal spikes, which simply means finding the sparse bit ones at each clock cycle from an input collection of bit ones and zeros. We coin this function as *unary top k*. As shown in Fig. 4a, existing SRM0-RNL neurons employ large PCs for accumulating response functions of input dendrites, requiring $n-1$ full adders for n inputs [7]. Our Catwalk neuron simply replace the large PC in the dendrite with unary top-k and smaller PC, with no other changes to the soma and axon. As TNNs integrate multiple SRM0-RNL neurons into one TNN column [7], [12], [13], Catwalk is a plug-and-play component that contributes to overall improvements in TNN efficiency.

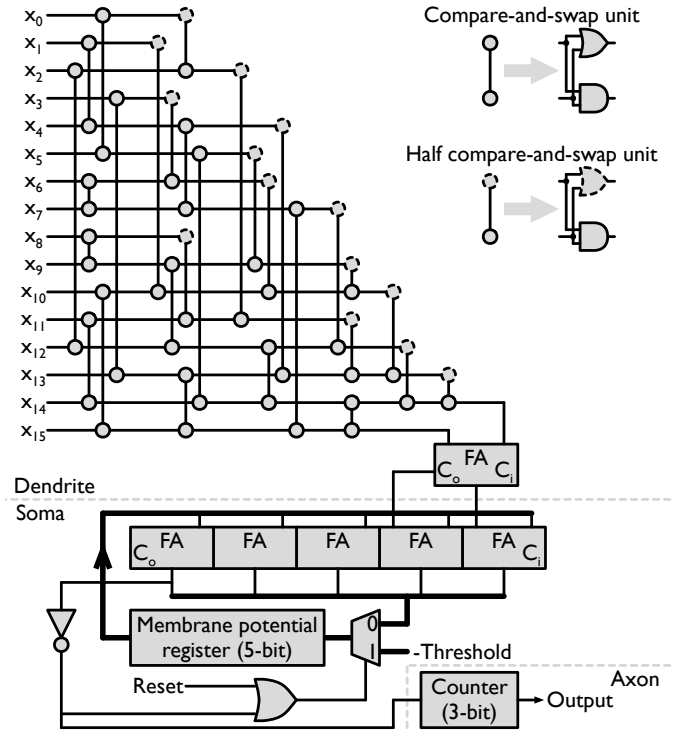
B. *Unary Top-K*

To implement unary top-k, we start with unary sorting in Fig. 3. Unary sorting has all inputs sorted and is very straightforward to find the top-k of all inputs. We show a few examples in Fig. 5, assuming that the outputs are in an ascending order from top to bottom. We evaluate two types of unary sorters: bitonic and optimal [2]. Bitonic sorters follow a structured bitonic pattern, while optimal sorters minimize the number of compare-and-swap units, achieving the lowest known count. We use Algorithm 1 to prune a given unary sorter and obtain the corresponding unary top-k selector.

According to Fig. 5, we have the following three observations. First different unary sorters can yield identical top-k results with varying pruning efficiency; for top-2, bitonic and optimal sorters prune equally, but for top-4, bitonic prunes more. Second, the final cost of unary top-k is independent of the cost reduction in compare-and-swap units. For both top-2 and top-4, the final costs depend on both the original cost



(a) Microarchitecture of an existing SRM0-RNL neuron. This example uses a 16-input PC as the dendrite to accumulate all possible input spikes (reproduced from [7]). The spikes are accumulated to the membrane potential register in the soma, which is then compared to a threshold to determine whether an output spike shall fire. The counter in the axon produces an 8-cycle pulse if an output spike occurs.



(b) Microarchitecture of our proposed Catwalk neuron. This example feeds on 16 inputs and selects top-2 outputs. Instead of a large PC, the dendrite is now implemented using unary top-k, based on the compare-and-swap units (Fig. 3b) and half compare-and-swap units. The half compare-and-swap units does not have the dash gate, whose output is no longer needed. The design of soma and axon remains identical to that in existing SRM0-RNL neurons (Fig. 4a).

Fig. 4: Microarchitecture comparison of existing SRM0-RNL neuron and our proposed Catwalk neuron.

Algorithm 1: Top-k pruning.

Input: $\mathcal{S}$: a list of tuples for a unary sorter, with each tuple representing a compare-and-swap unit and ordered from left to right; n: number of inputs; k: number of valid outputs.

Output: $\mathcal{T}$: a list of tuples for a unary top-k selector; $\mathcal{H}$: a list of tuples for the corresponding half compare-and-swap units.

$\mathcal{M} = [n-k, ..., n-1]$; /* Initialize top k outputs */
for *(i, j) in reversed($\mathcal{S}$)* **do**
  if *i or j in* $\mathcal{M}$ **then**
    $\mathcal{T}$.insert((i, j)); /* Insert tuple to front */
    $\mathcal{M}$.append(j or i); /* Add the missing j or i */
  end
end
$\mathcal{L} = \mathcal{T} + [(n-k, n-k+1), ..., (n-2, n-1)]$; /* Initialize tuple list */
for *(i, j) in* $\mathcal{L}$ **do**
  if *i or j not in remainder_node($\mathcal{L}$)* **then**
    $\mathcal{H}$.append((i, j)); /* Add half unit */
  end
end
return $\mathcal{T}$, $\mathcal{H}$

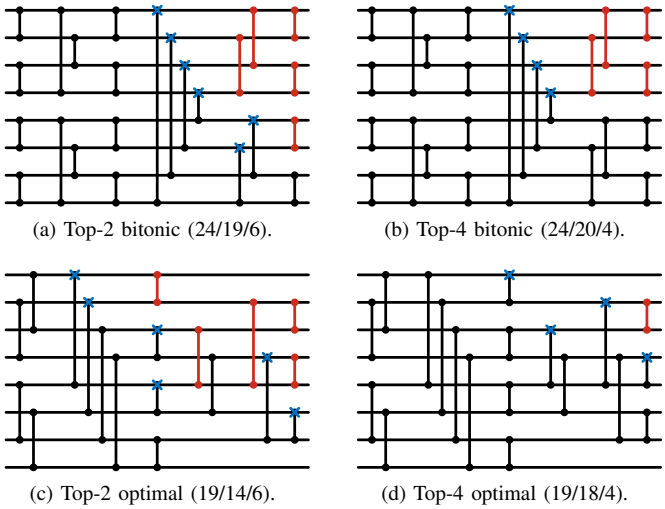
and the cost reduction. Third, the higher the k, the higher the hardware cost. From top-2 to top-4, less compare-and-swap units can be pruned, as well with the half compare-and-swap units (blue in Fig. 5).

In general, we observe that optimal sorters yield better results, so we choose them for this work. However, a gap remains between unary sorting and unary top-k, as directly selecting the top k without full sorting could be even more resource-efficient. This work focuses on optimal sorting-based solutions, leaving optimal top-k selection for future research.

V. Experimental Setup

We performed hardware evaluations using the NanGate45 standard cell library to obtain 45 nm CMOS implementation results and evaluation results. Design configurations of 16, 32, and 64 neuron inputs are chosen to enable comparisons across different scales. Synthesis carried out in Synopsys Design Compiler for three distinct design hierarchy configurations: (i) a stand-alone sorting/top-k stage, including unary bitonic sorters and optimal unary top-k, (ii) a sorting/top-k stage interfaced with a PC (a conventional design and a compact design), and (iii) bitonic sorting/optimal top-2 stage interfaced with a PC and augmented with a thresholding and firing unit, representing a SRM0-RNL neuron. At the final design hierarchy, all configurations are clocked at 400 MHz to ensure consistent timing assumptions.

Designs are then placed and routed using Cadence Innovus with the NanGate45 cell library. For these experiments, we again clock the designs at 400 MHz using a square floor plan with 70% utilization for each input size to provide a consistent basis for comparison.



(a) Top-2 bitonic (24/19/6). (b) Top-4 bitonic (24/20/4).

(c) Top-2 optimal (19/14/6). (d) Top-4 optimal (19/18/4).

Fig. 5: Comparison of unary top-k selector derived from different unary sorters with 8 inputs. The top-k outputs are at the bottom. (a) and (b) are pruning bitonic sorters, while (c) and (d) are pruning optimal, thus smallest sorters [2]. Each vertical segment represents one compare-and-swap unit in Fig. 3b. Black and red segments mark the mandatory and redundant compare-and-swap units in the unary sorter to implement unary top-k selector. Therefore, red units can be removed. Blue crosses mark these mandatory compare-and-swap units where only half of each unit is needed, i.e., one of the two outputs will not be used anymore. Blue crosses correspond to the dashed gates in Fig. 4b. x/y/z represent the number of total, mandatory, half compare-and-swap units.

VI. Evaluation

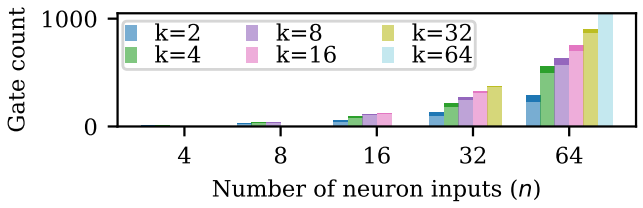
This section evaluates the cost of our proposed Catwalk neuron from both theoretical and experimental aspects.

A. Gate Count Analysis

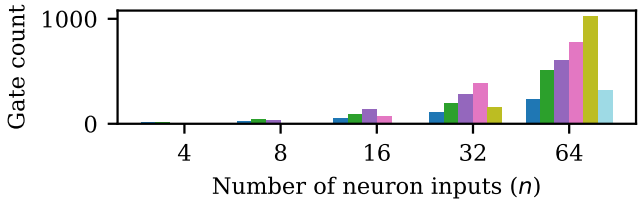
Given the crux of using a lightweight unary top-k selector and PC to replace the original large PC, and the corresponding large design space, we seek to understand the potential of Catwalk. More specifically, we examine the gate count of multiple unary top-k and dendrite input designs. We assume power-of-2 values for all n (inputs) and k (top-k selections).

Fig. 6a shows the gate count of different unary top-k designs. We use optimal sorters to derive these unary top-k selectors. The combined stacked bar is the total gate count for the remaining compare-and-swap units after pruning.

We observe that pruning compare-and-swap units significantly reduces hardware costs. Additionally, removing gates from half compare-and-swap units provides a smaller, but still helpful, optimization. The gate saving trends for $n = \{16, 32, 64\}$ demonstrates the potential of unary top-k in reducing the dendrite cost, thus neuron cost, from a theoretical aspect. Fig. 6b shows the gate count for different dendrite



(a) Gate count of unary top-k using Algorithm 1. Light color at the bottom is for the number of effective gates that contribute to the functionality, and solid color at the top is for the removed gate in half compare-and swap units. When $n == k$, unary top-k becomes unary sorting with no pruning.



(b) Gate count of dendrite. The dendrite adopts unary top-k (using Algorithm 1) and compact PC ($n-1$ full adders for n inputs). When $n == k$, the dendrite is just a large n-input compact PC without unary top-k.

Fig. 6: Gate count analysis of unary top-k and dendrite.

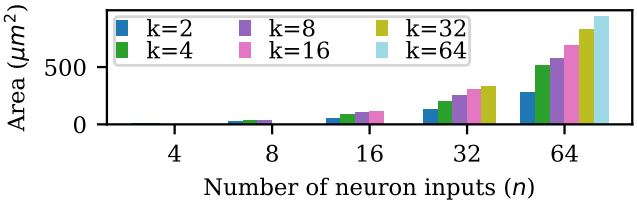
designs. We observe that when $k = 2$, unary top-k offers gains in gate count, while larger k values do not. This is due to unary sorting-based unary top-k becomes more costly with larger k, as shown in Fig. 5.

B. Synthesis Results

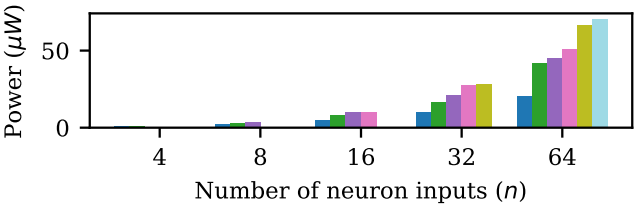
1) Unary Top-K: We synthesize unary sorting and unary top-k based on optimal sorting, which has the minimum number of compare-and-swap units known to date [2]. Only $n = \{4, 8, 16, 32, 64\}$ are publicly available, and we leave the exploration of larger n to future work. The unary top-k selectors are obtained using the same method as in Fig. 5. We show the results in Fig. 7. Fig. 7a and Fig. 7b show the synthesized area and power for different n and k, and we observe graceful scaling when sweeping n and k.

2) Dendrite: We also synthesize different dendrites, with results given in Fig. 8. To avoid an exploding design space, we do not explore all dendrite combinations, but focus on the most efficient and available options. As the sparsity in neurons can be as low as 0.1% [10], [11], [20], we consider $k = 2$ sufficient for the input count $n = \{16, 32, 64\}$. We do not consider larger n, as no such optimal sorters are publicly available [2]. Note that with $k = 2$, the PC for top-k and sorting is just one full adder, as shown in Fig. 4b.

Three observations can be made. First, unary top-k offers up to $1.17\times$ area savings over two PCs, aligning with theoretical gate count analysis. However, unary sorting does not consistently show better or worse area. Second, conventional PC (using an adder tree for accumulation) does not show worse area and power compared to compact PC (Fig. 4a). A conventional PC should have a larger cost in theory, but it is reasonable in the small scale that we are focusing on. Third,

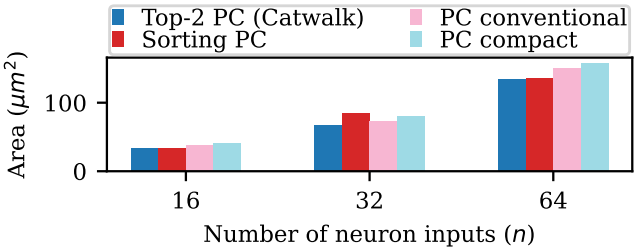


(a) Area.

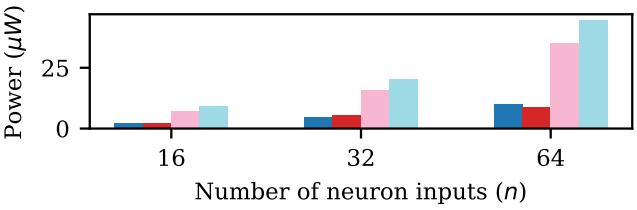


(b) Power.

Fig. 7: Synthesis results of unary top-k. When $n == k$, unary top-k becomes unary sorting.



(a) Area.

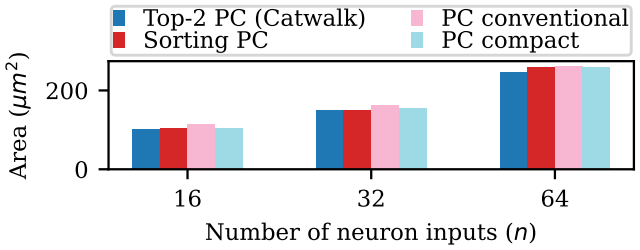


(b) Power.

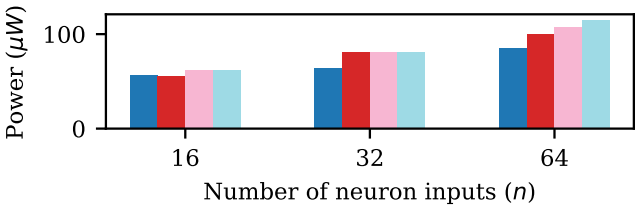
Fig. 8: Synthesis results of dendrite. Top-k here uses optimal sorters, while sorting use bitonic sorters. $n = \{16, 32, 64\}$ are studied, and k is fixed to 2.

both top-k and sorting show significant reduction in power consumption. The leakage power of different designs remains similar, while top-k (and sorting) lowers the dynamic power significantly, boosting the power efficiency by up to $4.52\times$.

3) Neuron: We further synthesize the full neuron, including dendrite, soma, and axon, with results given in Fig. 9. The experimental setup here is identical to that in dendrite evaluation. Catwalk (Top-2 PC) improves area and power by $1.05\times$ and $1.35\times$ over the neuron with a compact PC, and by $1.05\times$ and $1.17\times$ over the sorting-based neuron. This aligns



(a) Area.



(b) Power.

Fig. 9: Synthesis results of neuron. These neurons apply identical 5-bit accumulation and threshold implementation. Top-k here uses optimal sorters, while sorting use bitonic sorters. $n = \{16, 32, 64\}$ are studied, and k is fixed to 2. Note that the PC compact neuron is from [7], and the top-k PC neuron is our Catwalk neuron.

with insights from dendrite evaluation: while area reduction is limited, power improvement is significant.

C. Place and Route Results

In addition to synthesis, we also place and route all neurons, with the results in Table I. We observe that the leakage power of different design does not change too much, and Catwalk's benefits mainly origins from the reduction in dynamic power. Compared with the compact PC-based neuron (existing SRM0-RNL neuron design [7]), the area of Catwalk is improved by $1.23\times$, $1.32\times$ and 1.39; the power of Catwalk is improved by $1.38\times$, $1.67\times$ and $1.86\times$ for $n = 16, 32, 64$. In general, more improvements are present with larger n. Then the improvements are slightly more compared to the conventional PC-based neuron. Also, within the evaluated range, Catwalk indeed shows both area and power improvements compared to sorting PC-based neurons, indicating the importance of opting for top-k over sorting, despite identical functionality.

VII. Conclusion

This work identifies suboptimal efficiency of ramp-no-leak neurons in temporal neural networks, due to worse-case spike aggregation. We propose to relocate the temporal spikes via unary top-k to reduce the hardware cost. Unary top-k can be efficiently derived from unary sorting, with potential to reduce dendritic costs in neurons. Through the use of lightweight unary top-k and parallel counter, we show that the post place-and-route area and power can be improved by up to $1.39\times$ and $1.86\times$, respectively, compared to existing neurons.

TABLE I: Place-and-route results of different neurons in 45 nm CMOS. Neuron configurations are same as in Fig. 9.

Neuron design	**Power** (μW)			**Area**
	Leakage	**Dynamic**	**Total**	(μm^2)
$n = 16, k = 2$				
PC conventional	5.11	94.65	99.76	245.25
PC compact [7]	4.84	96.95	101.80	239.13
Sorting PC	4.28	70.11	74.39	197.64
Top-k PC (Catwalk)	4.22	69.40	73.62	194.98
$n = 32, k = 2$				
PC conventional	6.73	138.08	144.81	338.62
PC compact [7]	6.59	147.57	154.16	333.56
Sorting PC	5.73	88.24	93.97	256.42
Top-k PC (Catwalk)	5.66	86.79	92.45	252.97
$n = 64, k = 2$				
PC conventional	9.39	210.79	220.19	500.88
PC compact [7]	9.29	236.20	245.50	495.03
Sorting PC	8.12	129.59	137.71	364.15
Top-k PC (Catwalk)	7.85	124.21	132.06	355.38

References

[1] Shreyas Chaudhari et al. Unsupervised Clustering of Time Series Signals Using Neuromorphic Energy-Efficient Temporal Neural Networks. In *ICASSP*, 2021.

[2] Bert Dobbelaere. Smallest and Fastest Sorting Networks for A Given Number of Inputs, 2017. Accessed: 2025-03-06.

[3] Meng Dong et al. Unsupervised Speech Recognition Through Spike-Timing-Dependent Plasticity in A Convolutional Spiking Neural Network. *PloS one*, 2018.

[4] Robert Gütig and Haim Sompolinsky. The Tempotron: A Neuron That Learns Spike Timing–Based Decisions. *Nature neuroscience*, 2006.

[5] Saeed Reza Kheradpisheh et al. Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 2018.

[6] Wolfgang Maass. Networks of Spiking Neurons: The Third Generation of Neural Network Models. *Neural networks*, 1997.

[7] Harideep Nair et al. A Microarchitecture Implementation Framework for Online Learning with Temporal Neural Networks. In *ISVLSI*, 2021.

[8] Harideep Nair et al. TNN7: A Custom Macro Suite for Implementing Highly Optimized Designs of Neuromorphic TNNs. In *ISVLSI*, 2022.

[9] M. Hassan Najafi et al. Low-Cost Sorting Network Circuits Using Unary Processing. *TVLSI*, 2018.

[10] Rodrigo Quian Quiroga and Gabriel Kreiman. Measuring Sparseness in The Brain: Comment on Bowers (2009). *Psychological Review*, 2010.

[11] Shy Shoham et al. How Silent Is The Brain: Is There A "Dark Matter" Problem in Neuroscience? *Journal of Comparative Physiology A*, 2006.

[12] James E. Smith. A Neuromorphic Paradigm for Online Unsupervised Clustering. *arXiv*, 2020.

[13] James E. Smith. A Temporal Neural Network Architecture for Online Learning. *arXiv*, 2020.

[14] James E Smith. Implementing Online Reinforcement Learning with Temporal Neural Networks. *arXiv*, 2022.

[15] Amirhossein Tavanaei et al. Deep Learning in Spiking Neural Networks. *Neural networks*, 2019.

[16] Georgios Tzimpragos et al. Boosted Race Trees for Low Energy Classification. In *ASPLOS*, 2019.

[17] Prabhu Vellaisamy et al. TNNGen: Automated Design of Neuromorphic Sensory Processing Units for Time-Series Clustering. *TCAS-II*, 2024.

[18] Di Wu et al. uGEMM: Unary Computing Architecture for GEMM Applications. In *ISCA*, 2020.

[19] Di Wu and Joshua San Miguel. In-Stream Stochastic Division and Square Root via Correlation. In *DAC*, 2019.

[20] Amirreza Yousefzadeh et al. Asynchronous Spiking Neurons, the Natural Key to Exploit Temporal Sparsity. *JETCAS*, 2019.