Single Spike Artificial Neural Networks

Rhys Gretsch
ECE, UC Santa Barbara
Santa Barbara, California, USA
rhys@ucsb.edu

Michael Beyeler
CS, UC Santa Barbara
Santa Barbara, California, USA
michael.beyeler@psych.ucsb.edu

Jeremy Lau
CS, UC Santa Barbara
Santa Barbara, California, USA
lauj@ucsb.edu

Timothy Sherwood
CS, UC Santa Barbara
Santa Barbara, California, USA
sherwood@cs.ucsb.edu

Abstract

Spiking neural networks (SNNs) circumvent the need for large scale arithmetic using techniques inspired by biology. However, SNNs are designed with fundamentally different algorithms from ANNs, which have benefited from a rich history of theoretical advances and an increasingly mature software stack. In this paper we explore the potential of a new technique that lies between these two approaches, one that can leverage the software and system level optimizations of ANNs while utilizing biologically inspired circuits for energy efficient computation. The resulting hardware represents the traditional weight of an ANN as nothing more than a delay element and the degree of activation as nothing more than arrival time of a digital signal. Building on these fundamental operations, we can implement complete ANNs through several innovations: spatial and temporal reuse that facilitates classical dataflows, reducing memory system demands for ANN temporal operations; a new noise-tolerant temporal summation operation; novel hybrid digital/temporal memories; and the integration of temporal memory circuits for shepherding inter-layer activations. Using the MLPerf Tiny benchmark suite, we demonstrate how several architectural parameters can impact inference accuracy, that our proposed systolic array can provide 11× better energy consumption with a 4× improvement in latency compared to SNNs, and when equipped with temporal memories provides 3.5× improvements in energy compared to the most aggressive 8-bit digital systolic arrays.

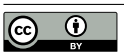
ACM Reference Format:
Rhys Gretsch, Michael Beyeler, Jeremy Lau, and Timothy Sherwood. 2025. Single Spike Artificial Neural Networks. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25), June 21–25, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3695053.3731027

1 Introduction

Time is an inherent property of all physical actions, one that is increasingly leveraged by designers. Devices that can sense, interact, and make decisions about the physical world around them are increasingly in demand [55], and one of the simplest ways to perform these tasks is by utilizing their temporal dynamics. Analog sensors first convert their value to temporal wavefronts, then to digital [67], while other sensors such as dynamic vision cameras [22] and lidar [59] already produce outputs with information encoded in time. This motivates a simple question: can we leverage these temporal dynamics without performing expensive time to digital conversions (TDCs)? This becomes an increasingly pressing question as we drive neural networks to energy constrained devices to create systems that not only develop an understanding of the world, but also make decisions based on that knowledge.

One approach for extracting these dynamics is spiking neural networks
temporal patterns from a series of spikes occurring across time. This is a seemingly completely different paradigm from ANNs (where linear layers of multiply-accumulate operations are interspersed with non-linear activation layers) and SNN's bio-inspired nature points towards the potential for incredibly energy efficient inference. The potential for energy efficiency comes largely from the data representation, in which spikes are encoded as binary values that facilitate lightweight, simple operations. However, this may also come at the expense of accuracy [50], latency [25], and design time [43]. Meanwhile, ANNs benefit from decades of research and industrial efforts creating mature tools and techniques that have shown to be effective across the computing stack, providing highly effective networks in terms of both accuracy and performance. In this work we propose a new approach that merges these seemingly distinct forms of inference, which can leverage the sparse, temporal coding of SNNs while still implementing the same algorithms underlying ANNs to achieve new levels of energy efficient inference. When coupled with traditional digital memories, this approach performs well compared to the most efficient and custom-trained SNNs, and when integrated with memristive temporal memories [37], will even beat 8-bit digital logic in terms of energy efficiency without any retraining or fine-tuning.

The key to our approach is the insight that we can encode each activation of the ANNs into just a single "spike" in time, relying on the temporal operations of "delay" and "real softmin" (operations that are both well defined and easily implementable on sets of spikes) to compute the linear operations that dominate ANN computation: multiplication and accumulation. Under this scheme, weights are applied to input spikes by simply delaying their propagation in time, as shown in Figure 1, which corresponds *exactly to multiplication.* A large weight in the neural network corresponds, very intuitively, with a small delay — with a zero weight equivalent to an infinite delay. Summation, under the same transformation,



ISCA '25, Tokyo, Japan

ACM ISBN 979-8-4007-1261-6/25/06
https://doi.org/10.1145/3695053.3731027

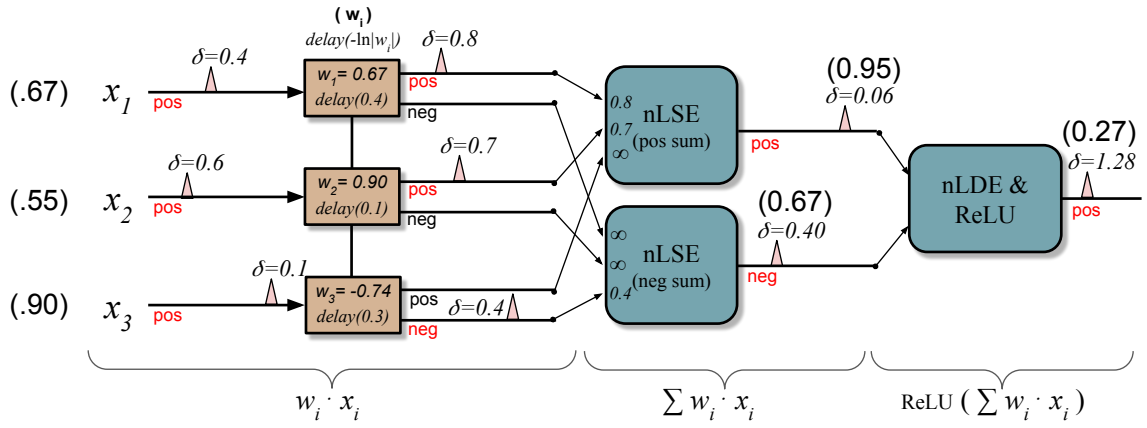


Figure 1: An example of a dual rail delay space neuron, with multiplication replaced by delays, accumulation performed with negative log sum exponential (nLSE) functions, and activation replaced by negative log sum difference (nLDE). The values on the far left show the original input and weight values before the delay space conversion. Each intermediate signal's delay space value is indicated with δ, with earlier values appearing towards the right. Higher importance values have lower delay and are thus first to arrive at downstream units. All weight delays are converted, quantized and stored digitally, with each weight's sign indicating whether to route the delayed value to a positive or negative rail (shown in red). Each value in the computation is represented by a single "spike" represented as a rising edge on a wire.

becomes an efficiently computable function using the primitive operations inherent to the temporal domain [35]. In this work we show how several important functions for ANN inference can be implemented using temporal primitives, creating a unified framework for implementing ANNs using a single spike per activation.

As one attempts to apply this arithmetic at larger scales, two new problems arise: a) prior methods of approximating real softmin show some critical non-idealities leading to error accumulation and b) internal digital memory becomes a greater and greater limiter. To address the first problem, we propose a noise-tolerant approximation method that can recover the accuracy of the original ANN through temporally interleaved min-term chaining. To address the second, we show the potential of temporal memories — memristive crossbars that have been demonstrated to capture and replay delayed signals — to bring the energy consumption below an 8-bit digital ANN. The resulting systolic array relies on programmable delay elements to provide a form of temporal quantization, while using temporal recurrence [10, 23] to maintain values in the temporal domain, reducing the number of memory accesses and facilitating classical dataflows. The specific contributions of this paper are:

- We describe how delay space arithmetic and approximations implement a spiking version of the linear operations in ANNs, then demonstrate how this method can be extended to efficiently implement ReLU and max-pooling.
- We are the first to evaluate how previously proposed delay space approximations impact neural network accuracy, and propose a new approximation methodology that provides improved network accuracy and better noise resiliency.
- We propose a hybrid temporal/digital systolic array that allows for weights to be loaded digitally, yet perform all operations in the temporal domain. Then we analyze how different architectural parameters impact network accuracy.

Importance	Delay Space
$\mathbf{x}$	$-\ln(\mathbf{x}) = x$
$\mathbf{w} \cdot \mathbf{x}$	$-\ln(\mathbf{w} \cdot \mathbf{x}) = -\ln(\mathbf{w}) + -\ln(\mathbf{x}) = w + x$
$\mathbf{x} + \mathbf{y}$	$-\ln(e^{-\ln(\mathbf{x})} + e^{-\ln(\mathbf{y})}) = -\ln(e^x + e^y) = \text{nLSE}(x, y)$
$\mathbf{x} - \mathbf{y}$	$-\ln(e^{-\ln(\mathbf{x})} - e^{-\ln(\mathbf{y})}) = -\ln(e^x - e^y) = \text{nLDE}(x, y)$

Table 1: Traditional "importance space" operations and what they correspond to in delay space. At any point a spike delay of x from reference time can be interpreted as an importance of $\mathbf{x} = e^{-x}$.

- We demonstrate how our systolic array can directly integrate with temporal memory systems, creating a system that consumes 3.5× less energy than a digital systolic array.

2 Neural Networks with Delay Space

SNNs attempt to model the relationship between presynaptic and postsynaptic spikes in a way that is biologically plausible. This is generally done with a leaky-integrate-and-fire model [9], which integrates incoming spikes and fires after reaching an activation threshold. These spiking models exhibit a variety of behaviors and may be abstracted further based on how information is encoded into the spikes — typically into one of two general categories, rate-coded and temporal-coded.

Temporal coding has inspired a general, logically complete algebra with four primary operations: min (first arrival), max (last arrival), delay (addition of a constant), and inhibit [51]. These operations have been shown to be incredibly energy-efficient when each "spike" is encoded as the rising edge of a voltage [35], giving each operation a simple logical equivalent (e.g. min corresponds to

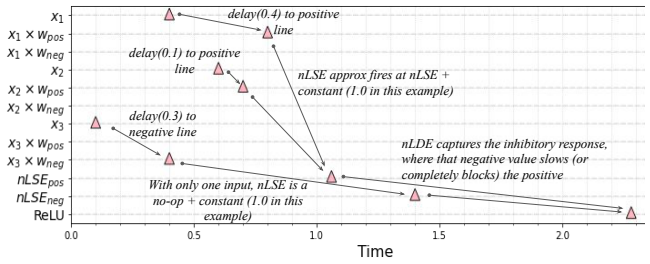


Figure 2: Graph showing the time of arrival and arithmetic dependencies for temporal spikes with the same example data from Figure 1. This additionally shows the constant offset required by the nLSE operation.

an AND). This temporal coding has been shown to be effective in decision trees [56], but there remains a large gap between the current capabilities of temporal computing and modern neural networks.

Delay space arithmetic [23] is based on this logically complete algebra, but relies on a negative natural log of the numbers and operations. Since the logarithmic function is undefined for negative numbers it necessitates a dual rail encoding to represent the complete number line. A value $\mathbf{x}$ is split and becomes $\mathbf{x}^{\text{pos}}, \mathbf{x}^{\text{neg}}$, where $\mathbf{x}^{\text{pos}} - \mathbf{x}^{\text{neg}} = \mathbf{x}$, $\max(\mathbf{x}^{\text{pos}}, \mathbf{x}^{\text{neg}}) = |\mathbf{x}|$, and $\min(\mathbf{x}^{\text{pos}}, \mathbf{x}^{\text{neg}}) = 0$. This pos/neg split cannot be directly converted to delay space, as one will always be zero, and the log of zero is undefined. To simplify the transition we define the delay space equivalent of zero to be infinity instead of undefined. Infinity in temporal coding is the same as a wire that never transitions from low to high. This means one of x^{pos} or x^{neg} will always be an inactive wire.

This transformation creates a natural fit for the four primary operations of temporal computing. Table 1 details how standard arithmetic, or "importance space", operations become transformed in the new negative log domain. Multiplication in importance space becomes addition in delay space, which can be implemented temporally with a simple delay. This is shown on the first functional block on the left side of Figure 1. With an assumption of positive only inputs (which we discuss in Section 2.1), this delay only needs to be coupled with a router to a positive or negative rail, depending on the sign of the weight. Next, addition in importance space becomes the negative log sum exponential (nLSE) function. This function, known as real-soft min can be approximated by a min-of-maxes approach, efficiently implemented using delays and basic logic gates, shown in Figure 3. The key technique that allows this to work is adding a constant offset to the result of the computation to avoid having to trigger an event before any inputs arrive, demonstrated in Figure 2. This accumulation in a dual rail system is shown as the middle functional block of Figure 1.

The result of this multiplication and accumulation will be maintained with two rails, as the negative and positive rails cannot be accumulated using the same function. However, these two rails both carry a value, breaking our original definition of dual rail coding. In importance space a simple subtraction can normalize these values, which gets transformed in the negative log domain to negative log difference exponential (nLDE). nLDE can be approximated using a min-of-inhibits, similar to the nLSE approximation. However, this function, like the original negative log transformation, is undefined for negative values. Instead it must be applied twice, with the inputs reversed:

$$y^{\text{pos}} = nLDE(x^{\text{pos}}, x^{\text{neg}})$$
$$y^{\text{neg}} = nLDE(x^{\text{neg}}, x^{\text{pos}})$$

The approximation returns ∞ when nLDE is undefined, so the result of these two operations will always adhere to the original definition of the dual rail coding. In order to maintain this strict dual rail definition this normalization must be applied after every operation. However, the associative nature of the preceding linear operations (additions and multiplication) apply to their log transformation (nLSE and addition), so the nLDE only needs to be applied before non-linear operations.

These three transformed operations (delay, nLSE, and nLDE) *exactly* implement the linear operations within neural networks.

2.1 Non-Linear Temporal Operations

Unlike prior work, we show this framework can be extended to several other operations to support the non-linear activations necessary for ANNs, an integral part of neural networks. The first important operation for energy-efficient neural networks is ReLU, where only positive values are passed through and all negative values become zero. In delay space this zero becomes infinity, or a wire that never triggers. Therefore, to implement ReLU, only the positive rail of the previous computation needs to be calculated. Due to the inhibit function used in the nLDE approximation, the required infinity is the natural result when the result would be negative in importance space. This makes the min-of-inhibits approximation a natural fit, as digital implementations of nLDE would return undefined or NaN.

The second operation necessary in many convolutional neural networks (CNNs) is max-pooling. If one of the values in the max-pool operation is positive, then a simple min of the positive rails can implement max-pooling. This max to min conversion stems from the negative log transform, where larger values in importance space are transformed to smaller delays in delay space. Alternatively, if none of the values are positive, then a max of the negative rails is used. This can be simplified to:

$$y^{\text{pos}} = \min(x_0^{\text{pos}}, ..., x_n^{\text{pos}})$$
$$y^{\text{neg}} = \max(x_0^{\text{neg}}, ..., x_n^{\text{neg}})$$

The simplification of this logic works because any positive value will have an infinite negative rail, causing the max of all negative values to never trigger. For networks where ReLU is combined with max pooling, the operation becomes even more simple. The only relevant information is the positive rail and is performed simply with a min operation. These two operations are sufficient to implement many neural networks [3], and pre-trained networks can easily be converted to delay space by simply transforming all inputs and weights to delay space values. For the weights of a network, this represents a one-time cost that can be performed before loading them onto the system. For the inputs, this step can

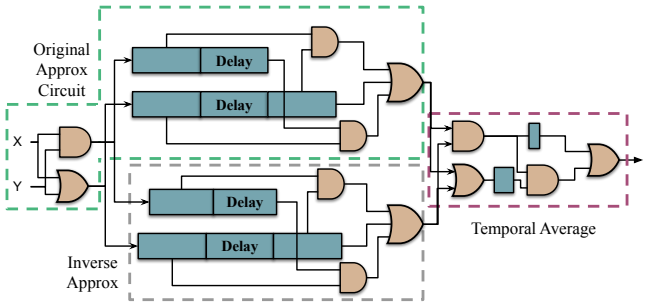


Figure 3: Hardware diagram of the improved nLSE approximation circuit, relying on redundancy to provide better accuracy and noise characteristics.

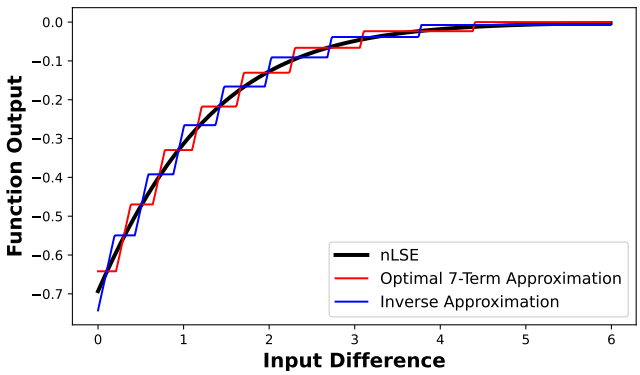


Figure 4: Function graph of nLSE and 2 complementary min-of-max methods of approximation.

be co-located with the input pre-processing, and only needs to be performed once before the network begins, representing a tiny fraction of the overall computational demand of the network.

2.2 Noise Resilience

We've shown that we can algorithmically merge ANNs with temporal computation, however, while temporal computation relies on digital tools and building blocks, the domain is inherently noisy. Further, the log transformation that enables temporal arithmetic introduces logarithmic bias, where balanced error and noise in delay space become skewed in importance space. In Section 6.1, we show how this degradation requires significant area and timing overheads when naively applying delay space approximations to neural networks. To reduce the impact of both the noisy temporal operations and the approximation bias, we discuss two techniques. First, we propose a novel, circuit level optimization of the nLSE approximation that reduces logarithmic bias while simultaneously being more resilient to noise. Then, we discuss how standard regularization techniques create networks that are more robust to the types of noise introduced by our temporal networks.

Improved approximations: Approximations of delay space addition (nLSE) rely on a min-of-maxes approach, where every max-term creates a deformation, or "valley", in the original min function. These valleys create lines with positive and negative unit slopes, and careful placement of these valleys allows for the non-linear function nLSE to be approximated. Additionally, this approach only needs to approximate half of the function's range by operating on the difference between the inputs and adding the results to the minimum input. This is a common trick when computing real soft min and max, and can be implemented in hardware using a simple temporal routing scheme shown on the left side of Figure 3. This changes the function being approximated to $\text{nLSE}(0, \max(x, y) - \min(x, y))$, which is shown as the black line in Figure 4.

The red line in Figure 4 shows a 7-term approximation using the same optimization method as [23] to place each valley. This method attempts to minimize the distance between the approximation and the original function by balancing the area above and below the original curve. However, as we show in Section 6.1, this approach struggles to fully recover the accuracy of convolutional neural networks. This is largely because error that's balanced in log space becomes unbalanced when converted back to importance space, introducing an inherent bias to the approximations.

Our proposed solution to this bias relies on creating an equal, but complementary approximation to the original, shown in Figure 4 as the blue line. This approximation uses the same optimization framework as the prior work [23], but starts the solver using a valley instead of a peak. Taking the average of these two approximations provides a near-exact copy of the original nLSE function. Unfortunately, implementing an *exact* temporal average would require analog circuitry, preventing the temporal logic from being implemented with purely digital building blocks. Instead, an approximate averaging circuit can be built by characterizing the differences between the approximations.

This approximate average circuit is developed by using the delay space equivalent to the average of two values: $\frac{\mathbf{x}+\mathbf{y}}{2}$ becomes $\text{nLSE}(x, y) + \ln 2$. This allows the same nLSE approximation technique to be used, with all constants shifted by $\ln 2$. However, the characterization of the two inverse approximations provides a tight bound on the difference between the approximations at any given point along the curve. Now only a small portion of the function's curve needs to be approximated, allowing an acceptable approximation with minimal additional delay and gates.

This combined approximation, shown in Figure 3, creates a better piecewise approximation of the original nLSE function, providing better accuracy than the original approach. Temporally combined approximations additionally provide redundancy, allowing the resulting value to be less sensitive to noise along either of the lines.

Neural Network Regularization: Regularization is commonly used when training neural networks to prevent over-fitting. There's a broad range of techniques, weight decay [29], noisy layers [6], and dropout [52], which attempt to prevent networks from becoming overly reliant on certain features of the input data. We show in Section 6.1 how networks trained with regularization are much more robust to the error introduced by our temporal networks, particularly when combined with our improved approximation. While this must be applied at training time, it requires minimal modification to the training stack and most networks already include some form of regularization.

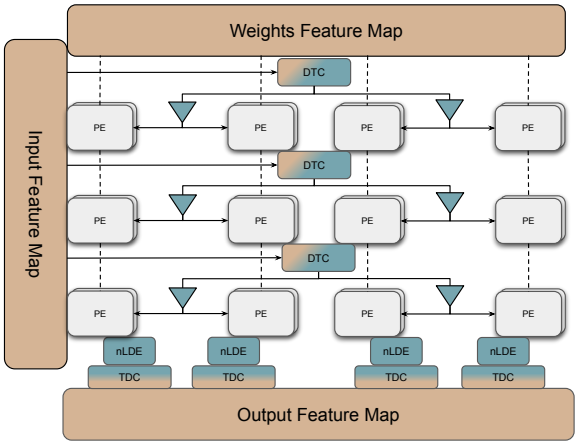


Figure 5: Architecture of the hybrid temporal/digital systolic array where all memory is digital, so accesses to the input and output feature maps require temporal conversions, either temporal to digital (TDC) or digital to temporal (DTC). The DTC is placed in the middle of the array to balance the broadcast path to each PE.

3 A Temporal/Digital Systolic Architecture

While the proposed techniques shown in Section 2 demonstrate how artificial neural networks can be mapped to a temporal equivalent, prior implementations of temporal logic have either been fixed function [23, 35] or application specific [56]. This represents a significant gap between our proposed delay space networks and both ANNs and SNNs, as the ability to program new weights and biases is at the heart of these accelerators. To bridge this gap, we propose a novel hybrid temporal/digital systolic array to support the general class of delay space networks shown in Figure 5. Due to the algorithmic equivalence to ANNs, this systolic array leverages many of the same patterns as digital systolic arrays, exploiting reuse wherever possible and only falling back to digital when necessary. However, the transient nature of temporal computation makes many digital approaches to achieving these reuse patterns ineffective, as they require numerous domain conversions, which counteract the benefits of delay-space computation.

3.1 General and Programmable Operation

Programmable neural network accelerators begin with the ability to program weights, typically from memory into a register, which can be directly manipulated by logic. However, our transformed weights are encoded as delays, necessitating a circuit or device that has a variable propagation speed based on some external, programmable input. Several programmable delays have been proposed, including starved inverters [60], memristors [37], and pull-down networks [39]. These generally work by artificially increasing the resistance and capacitance within a single device or gate, attempting to create a linear relationship between the programming signal and the corresponding delay. Starved inverters and memristors can achieve this linearity, but are controlled through analog signals: steady state voltages and voltage pulses respectively, which would require another domain conversion step. Pull-down networks are digitally programmable, but offer a narrow range and high delay variability [38].

As an alternative, we use a cascading inverter approach [13], where each bit programs a mux that selects between a delay path and an immediate path, as shown in Figure 6a. Each mux controls a delay path that is twice as long as the previous stage, providing 2^n delay paths. This leverages the fact that delay circuits use the same binary wires as digital logic, allowing digital control to direct and route temporal paths. However, this does limit the possible number of delay paths, forcing a fixed-point quantization.

Quantization, a technique that reduces the bit-precision of operations [42], is quintessential to most neural network accelerators, leveraging ANN's tolerance to error. However, due to the low dynamic range of these quantized values, it's often necessary to use scaling values to maintain network accuracy [63]. This introduces multiplication and rounding overheads that are typically a small fraction of the energy saved, but require additional, higher precision hardware units. Delay space quantization must use a similar scaling scheme, partially to ensure all temporal values are positive. However, since delay space networks use a log number system, the multiplicative scaling can be simplified to addition. Additionally, the log number system allows for a wider dynamic range than an equivalent number of bits in a linear system, and is thus a promising candidate for many neural network applications [40, 68]

While a programmable delay is sufficient to create a general purpose architecture, without leveraging reuse, both spatially and temporally, the memory system will dominate the energy consumption of the system. Typically, when inputs are read from SRAM, classical architectures exploit spatial reuse through a store-and-forward method [47], allowing values to be used by processing elements (PEs) across space and time. However, this technique relies on registers, a structure that doesn't have a clear temporal equivalent. As we show in Section 3.3, memristive temporal memories could potentially act as a register, but the control logic makes it inefficient to implement as standalone devices inside the systolic array. Alternatively, the temporal signal can be converted to digital, stored in a register, then replayed on the next cycle. However, this domain conversion, particularly TDCs, is both area and energy intensive, and should be minimized wherever possible. Instead, we broadcast the temporal signals to all the PEs within a row to achieve the same level of spatial reuse as the store-and-forward approach. This broadcast can be achieved by placing the DTC physically in the center of the array. From the DTC, a branching routing scheme can be used to ensure that each PE receives the signal at the same time, similar to clock network H-trees [64].

After each PE's outputs are generated, most systolic arrays exploit temporal reuse by storing and replaying the result across subsequent cycles. However, the output of our delay space multiply and accumulate (MAC) unit is inherently temporal and again, cannot be stored effectively as-is. It could be immediately converted to digital, similar to methods used by other hybrid domain accelerators [62], but this conversion is expensive and doubles the energy consumed by each PE. Instead, we propose maintaining the temporal outputs by resynchronizing them with the next cycle's incoming data using temporal recurrence [10, 23].

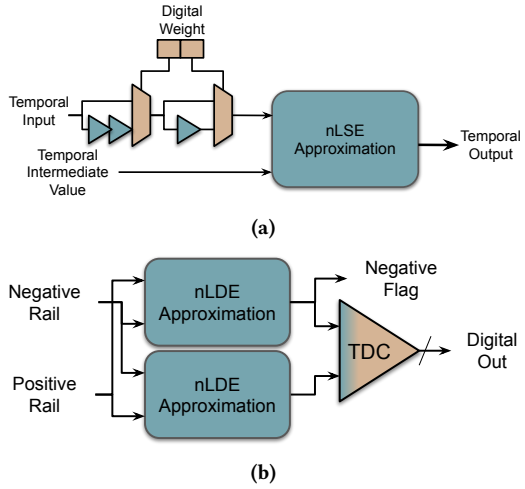


Figure 6: a) The delay space MAC, equipped with a digitally programmable delay element. b) Paired nLDE-TDC readout with the automatically generated negative flag.

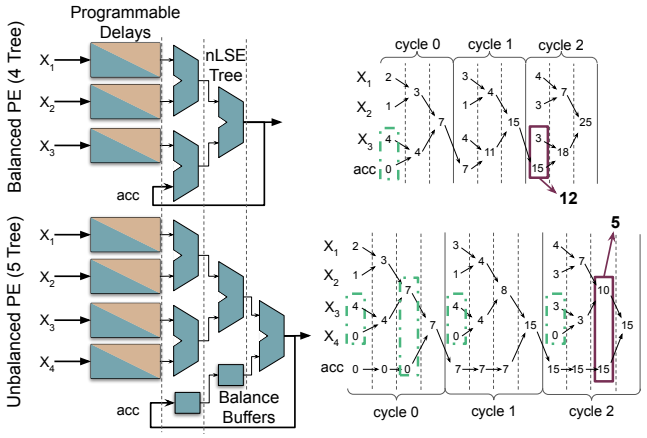


Figure 7: Two possible types of PE tree sizes, balanced and unbalanced (left) and their corresponding execution of an importance space set of data (right). We highlight the areas where the approximation is perfect with the dotted green boxes, and the maximum difference that needs to be approximated in the solid boxes. This max difference exemplifies how the PE structure impacts the nLSE input distribution that needs to be approximated.

This works by delaying the output of the signal so that it has the same scale as the next input, synchronizing the two signals. For instance, if two delay space values x and y arrive across two consecutive cycles, the actual arrival times are t_x and $t_y + T$ where T is the cycle time. However, addition in time is the same as importance space multiplication, so the $+T$ term acts as a scaling factor. To operate on the two values, the same scaling factor must be applied to both, which can be accomplished by delaying x by the cycle time. Also, it's important that any operations performed this way must be distributive so that $f(x+T, y+T) = f(x,y) + T$, a requirement met by all delay space operations. While this does require delay to perform the synchronization, it can be combined with the delay inherent to the nLSE approximation. Since this approximation is along the critical path, minimal additional hardware is necessary to support recurrence.

This resynchronization along with temporal broadcasting exploits spatial and temporal reuse in systolic arrays, allowing the delay space spikes to leverage many of the same systems set up for ANNs. This reuse, alongside the digitally programmable delay elements, creates a general purpose architecture that can map all of the linear layers of a delay space network, and by extension all of the linear layers of an ANN.

3.2 Error Minimization through Temporal Logic Trees

In digital logic, floating point values suffer from an accumulation of errors as more computation is performed [28], and the shape of the computation has a large impact. A tree based reduction of computing floating point values has much less error than a pure linear reduction approach. This same phenomenon is compounded in the delay space approximations, where error minimization is critical due to the log-compressed nature of delay space. This is caused by two reasons, stemming from the approximation technique. First, nLSE with a larger input difference than the range of approximation becomes the min operation. This does not introduce much error in isolation, but a linear reduction creates a very small value in delay space. Since the reduction cannot include negative numbers, this value eventually becomes much smaller than the original input domain. At this point, every nLSE becomes min, losing all information that would be provided by additional accumulations. The second impact comes with the uneven accuracy of the approximation. Certain points along the approximation curve are closer to the true nLSE function than others, and so the accuracy is dependent on the absolute difference of the two inputs. This distribution of inputs changes as the reduction changes from linear to tree based, which we explore in Section 6.1.

In digital systolic arrays, the parallel nature of data representation constrains PEs to accept only one input, one filter, and one intermediate value per cycle [12, 27], resulting in a fully linear reduction. However, multi-input PEs have been shown in unary computing systems [61] and aggressively quantized systems [4] because both the required bandwidth and wire congestion is significantly reduced. This same approach can be used with our proposed array as shown in Figure 7.

Because the delay space nLSE approximations are designed for approximating a two-input function, a hardware tree must be used to fully reduce all of the incoming values. For maximum accuracy a tree large enough to accept all of the values in parallel would be used, but the large number of input values per neuron in neural networks makes this infeasible. Instead, the tree size is fixed and the values are folded across time, creating a semi-serialized reduction approach. If the reduction tree is fully balanced, then no extra hardware is required, but an unbalanced tree requires extra synchronization lines due to the constant offset introduced by each stage in the nLSE tree. Also, the PE tree size creates a new trade-off between

performance, energy, and accuracy. The larger the depth of the tree, the longer each cycle takes while requiring more memory accesses to ensure the PE is fully utilized. We explore these tradeoffs in Section 6.

3.3 Temporal Memory

While our proposed array up until this point shows how to minimize TDCs and DTCs, these conversions are still necessary to interact with memory, a classically digital structure. However, recently there have been large strides in creating memory systems that can capture timing of wave fronts, which can then be recalled on demand [37, 44, 45, 57]. These typically rely on *memristive*[53] devices, which have a programmable resistance based on the duration of a write pulse, leading to a direct relationship between the physical characteristic (resistance) and time.

Programming these devices requires a temporal signal and a reference signal, which can be directly generated by the clock. The memristor is connected in parallel with a transistor between the word line (the reference signal) and the bit line (the temporal signal). The transistor is then programmed by a select line that determines whether that specific device is being activated. When a device is selected for a write operation, the transistor is turned on, and when the reference signal goes high it creates a voltage difference across the memristor, which in turn causes the device's internal resistance to increase. Once the temporal signal arrives, the voltage difference becomes zero, preventing any more changes to the memristors state. As long as the difference between the reference and temporal signal is less than 40ns, which delay space operates within, this change in resistance is linear.

During a read the word line goes high at a lower voltage (to prevent modifying the memristor's state), which causes current to flow across the memristor, charging a capacitor along the bitline. This capacitor can be connected to a gate that goes high once the voltage across the capacitor reaches a certain threshold. The time it takes to reach this threshold is linearly related to the resistance of the capacitor, causing the output pulse to be linearly related to the time between the reference pulse and the original programming signal. Using additional capacitors, this signal can create a one-to-one mapping of the programming and read signals. Although this operation introduces inherent delay, it is equivalent to a constant input reference shift.

This represents slightly more complex control circuitry than classical SRAMs, but its advantages far outweigh the cost. The input and output feature maps can be completely replaced with their temporal equivalent, totally removing the need for any TDCs or DTCs within the array. Additionally, a single device maintains a full value instead of 8 SRAM cells needed for every activation. This allows for a significant reduction in read/write parasitics, causing the memristive operations to dominate the energy consumption. While this is more expensive than a typical memory cell access, it still provides significant savings over it digital counterpart, as we analyze in section 6.2.

Precision of recall is the largest barrier to using this temporal memory. While this represents a challenge, memristive fabrication consistency has improved significantly, with device to device variability reaching less than 1% [30] and cycle to cycle variability small enough for 7 bits of precision [34] with current technologies. As memristor technology matures this variability will decrease, allowing for better distinction of stored wavefronts. While the variability will never completely go away, it will serve as a form of quantization, introducing error in a similar manner to the error introduced by temporal activations being quantized as they're stored digitally.

4 Dataflow for Temporal Operation

While Section 3.1 shows how reuse can be exploited at the micro-architectural level, it is well known that how this reuse is structured at the global level has a large impact on the energy of the full system for both ANNs and SNNs due to the orders of magnitude difference between the cost of a MAC and a memory access. ANNs must balance folding their inputs, weights and outputs across the systolic array and network layers, but SNNs typically have to balance all of these things across time as well [66]. This extra dimension increases the dataflow complexity and design space possibilities.

In general, because delay space performs the same operations as ANNs the array can leverage the simplicity of ANN dataflows. However, the spatial and temporal local reuse exploited by our architecture is based on the temporal values, which establishes limitations that prevent some dataflows. For example, dataflows that require inputs to be passed diagonally across the array are no longer effective because of the input broadcasting.

Also, many accelerators co-locate PEs with small scratchpads to maintain multiple sets of intermediate data to maximize utilization and reuse[33]. However, the recurrence that facilitates temporal reuse in our architecture incur increasing energy and noise as multiple cycle delays are required, making the flexible designs built on scratchpads impossible. Instead simpler data flows must be considered, like those used in systolic arrays that only allow for neighbor-to-neighbor communication. These data flows fall into three categories: weight stationary (WS), output stationary (OS), and input stationary (IS), and have been shown to be very effective [27].

IS requires that after each input arrives at a PE it is stored and reused every cycle in the same PE. Again, the temporal inputs to our systolic array make this challenging. The input could be delayed to re-synchronize with the next cycle to maintain the temporal representation as discussed in Section 3. However, the value that must be delayed is unaffected by the computation, so none of the recurrence delay is hidden by the computational delay. Instead there must be a long delay line to offset the signal by an entire cycle, consuming additional area and energy. Because of the incompatibility with the hybrid array we ignore IS and instead discuss how both WS and OS can be implemented using the array.

Implementing a WS dataflow is straightforward because the weights in the systolic array are fully digital. The programmable delay of each PE is latched to the incoming weight value and does not change until that weight has interacted with all necessary inputs. The temporal output of each PE is routed along the same column to the next PE where it is delayed to be synchronized with the input of the next cycle. This output is not being passed back into the same PE, no longer implementing recurrence. Instead, the same synchronization technique implements a temporal pipeline.

The OS approach only needs one PE to compute a single output pixel/value, while the necessary weights and inputs are streamed through the PE. The temporal result of each PE is synchronized with the next set of inputs, then fed back into the same PE that it came from, implementing recurrence exactly. Once the entire output is calculated, the temporal result is sent to be stored digitally. This output has been fully computed so only the activation function needs to be computed. For ReLU this means that half of the normalization nLDE circuits shown in Figure 6b can be removed, as only the positive rail is taken into account.

When considering a design that leverages temporal memories, the input and output feature maps are smaller and more energy efficient, while the weight feature map is still implemented digitally. Since this consumes much more energy per access compared to the other memories it's wise to try and minimize this access. Therefore, for the rest of this paper we leverage a WS dataflow to fully take advantage of the temporal memories. While this approach more frequently leverages the nLDEs at the interface to the memory, it's more than offset by the energy efficiency gained by the cheaper access. Additionally, this periodic partial sum normalization can prevent recurring values from growing too large (fast) compared to the input signals.

5 Evaluation Framework

We used three different frameworks to evaluate our proposed system. First, to evaluate the accuracy of delay space neural networks we created a simulation framework based on TensorFlow[2]. This approach takes a pre-trained neural network and dataset, then converts all of the weights and inputs into delay space values. Next, it takes each layer and modifies the layer so that all calculations performed are replaced with their delay space equivalent (e.g., multiplication becomes addition and addition becomes our nLSE approximation). We add an option to perform these operations with noise based on prior analysis [41]. Generally noise comes from three sources: process, voltage and temperature (PVT). Process variation is correlated across large parts of the chip, which acts as a constant shift to all temporal operations. Since our operations are all shift invariant, this variation has minimal impact on the result of the computation [13]. For temperature, or random jitter (RJ) we model delay elements with independent Gaussian noise, so the noise grows at a reduced rate as the number of delay elements increase. Our voltage variation model assumes a distribution where the limits are determined by the potential swing in supply voltage. This model assumes a worst case scenario where voltage variation is correlated within a delay line, but uncorrelated across both space and time, so every operation is exposed to potentially maximum noise. Power supply jitter would normally impact the array globally, exposing all circuits within a cycle to the same shift, causing impacts to only appear over time. However, this models a rapidly changing power supply to provide an extreme limit on power supply impact.

Next, to evaluate the energy efficiency, performance, and area of the delay space computations we use a 32nm technology model [69] to determine the energy consumption of the delay elements and basic operations in SPICE, using a 700mV supply voltage. We then use an analytical model to extend these values to determine the energy consumption of the delay space operations for a given unit scale, which determines how theoretical delay space values get mapped to physical timings (the unit scale is the physical delay that is interpreted as the value 1). For the TDC and DTC we use our SPICE modeling values and values from a simulated Vernier delay line [18]. To evaluate the energy efficiency of the full system we modify SCALE-Sim [46, 47], a systolic array architectural analysis tool. SCALE-Sim was modified to support the broadcast inputs of our proposed system and the multi-input PEs. This tool provides memory usage statistics, which we use in conjunction with Cacti 7.0 [7] to model the energy and area consumption of the digital memory systems. To evaluate our array in conjunction with temporal memory we rely on a 32nm RRAM process in NVSim [19], including the additional overheads of temporal reads and writes [37]. For the temporal memories analysis we include the TDC and DTC cost of all off-chip accesses that must interact with the temporal memories.

To evaluate our design we use the MLPerf Tiny benchmark suite[8], which provides neural networks for edge applications where ultra-low energy inference is necessary. In order to evaluate the impact of quantization on network accuracy we use a scale quantization method [26, 63] with no fine-tuning for the baseline importance space model, and use a transformed version of the same scheme for the delay space quantization. For a baseline and comparison we consider a digital systolic array and SATA, a state of the art spiking neural network accelerator[66]. The digital design is evaluated with computing units synthesized in Synopsys Design Compiler at 400MHz using a 45nm tech node [61], and SATA-sim to evaluate the SATA architecture. For both baselines we use tech node scaling [48] to scale both results to 32nm to ensure a fair comparison.

6 Results

6.1 Hardware Impact on Accuracy

The configuration of the hardware in the delay space framework has a significant impact on the accuracy of the transformed neural network, so it is important to understand how the different hardware parameters can impact the performance of the neural network.

The hardware parameter that has the largest impact on neural network accuracy is the number of approximation terms, and in Figure 8 we show how the number of max-terms for nLSE approximation impacts the accuracy of the MLPerf Tiny benchmarks. For each of these graphs we fix the number of inhibit-terms for the nLDE approximation to 25, which we've experimentally determined to give sufficient network accuracy. The x-axis shows the number of approximation terms, while the y-axis shows the performance metric. We show two networks for each application, one trained with dropout for regularization and one trained without dropout. We also compare against our proposed approximation and prior approximations. For comparison, we count our proposed approximation to have the same number of terms as two times the original approximation (e.g a 2×7 term approximation appears on the graph as 14 approximation terms). In addition, each graph has two lines that indicate the original network's accuracy and the performance target set by the MLPerf Tiny benchmark.

In the original model, trained without dropout, our proposed approximation method has better accuracy for any given number of terms, with the exception of low-term approximations of anomaly detection. For the visual wake word and keyword spotting benchmarks, a 2×3 term approximation is sufficient to reach the performance target, while the original approximation approach requires 15 terms to achieve the same target. This improvement is exemplified even further when applied to the regularized models. For the two networks already robust to noise, the original accuracy can be almost fully recovered with 2×5 terms, and 2×7 terms is sufficient to hit the performance target for every benchmark. While the original approximations see similar improvements from the use of regularization, it still takes 25 terms to reach the image recognition performance target.

However, this analysis assumes these approximations can be implemented perfectly in hardware, with full precision. Both quantization and hardware based noise will impact the final accuracy of the network, and need to be considered. To examine this impact we apply the noise models discussed in Section 5 to an 8-bit ResNet8 (the MLPerf Tiny image recognition network) trained with dropout. The results of the noise analysis on a 2×12 and 25 nLSE approximation term system are shown in Figure 9.

First we examine the impact of random jitter (RJ) on accuracy without voltage variation, shown by the blue curve and x-axis, as RJ is largely based on the device physics and can only be controlled by increasing the unit scale, which in turn increases the computation time. At large unit scales the accuracy degradation is minimal, and similar regardless of which approximation method is used. However, as the unit scale reaches 1ns and shorter, our proposed approximation accuracy decreases less than the original, staying above the performance target with a 750ps unit scale.

We then perform an experiment that considers both RJ and voltage variation, assuming a base 1ns unit scale for RJ. Voltage variation is swept, parameterized by the potential voltage swing as a percentage of the original supply, and shown in the red lines of Figure 9. This demonstrates how voltage based noise can have a much larger impact on network accuracy. However, as discussed in Section 5, this is assumes a system much noisier than any realistic hardware, with potentially maximal variation occurring multiple times every cycle. The primary takeaway is our improved approximation is less impacted by noise, even under extreme conditions.

These two experiments show at a circuit and logic level how the approximations can impact neural networks, but architectural design decisions also impact the overall accuracy, as discussed in Section 3.2. In Table 2 we show how 4 different PE tree configurations can impact accuracy (n-tree indicates a PE tree with n inputs), assuming a 2×12 nLSE approximation terms. We consider 3 balanced tree configurations, and one unbalanced configuration (9-tree).

While most of the benchmarks have minor accuracy fluctuations with changing PE tree sizes, anomaly detection degrades drastically. While the performance can be recovered with a large, unbalanced tree, balanced PEs have much worse accuracy. The reason for this degradation is shown in Figure 10, displaying how the approximation input distribution changes drastically with the tree size. When anomaly detection uses balanced tree PEs, it creates a much larger spread of possible input values than image recognition, exemplified

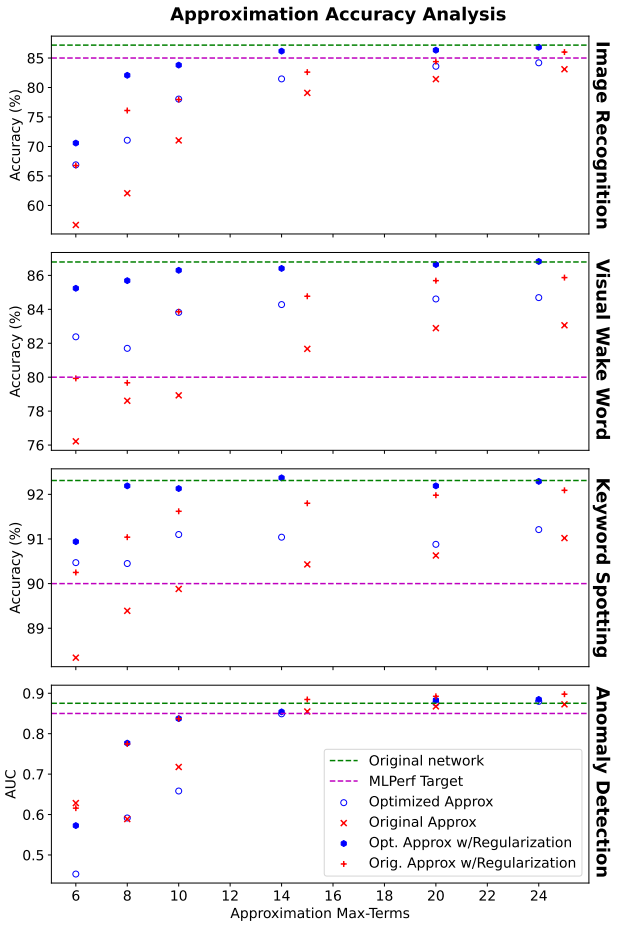


Figure 8: Network accuracy impact of the original delay space approximations (Single Approximation) and our optimized approximations (Dual Approximations) (10 max-terms for the original approximation correspond with 2×5 terms for our proposed optimized approximation). The number of inhibit-terms for delay space subtraction is kept constant at 25 terms for each network. We show the converted network accuracy both when the original network is trained with and without regularization. The top horizontal dashed line represents the original ANN accuracy, while the purple line represents the minimum performance required by the MLPerf guidelines.

Table 2: MLPerf Tiny benchmark accuracy as the PE tree size is swept, including one unbalanced tree configuration.

	9 Tree	8 Tree	4 Tree	2 Tree
Image Recognition	86.36%	85.72%	86.41%	85.65%
Visual Wake Word	86.57%	86.42%	86.65%	86.44%
Keyword Spotting	92.04%	92.72%	91.35%	91.21%
Anomaly Detection	.857	.48	.47	.46

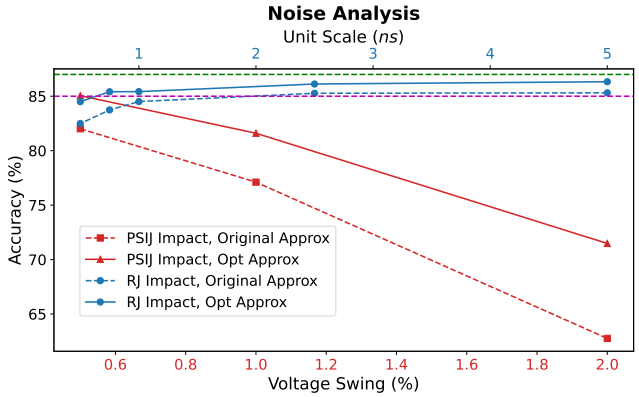


Figure 9: Analysis of how random jitter (RJ) and voltage variations impact the accuracy of delay space implementation of an 8-bit ResNet8 for image classification. The RJ analysis is performed without voltage variation and is shown in blue, while the voltage variation analysis includes RJ with a 1ns unit scale.

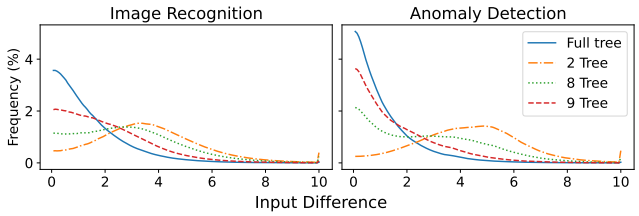


Figure 10: Histograms of the nLSE approximation input values for the image recognition and anomaly detection benchmarks as the PE tree size is changed.

by the differences in the 2 tree PE histograms and the wide flat region that occurs in the 8 tree. The approximation is the most accurate as values are close to one another, and less accurate as they move further apart, so this larger spread means more operations are being approximated poorly. Using a 9 tree PE anomaly detection moves the distribution significantly closer together, where the approximation is better, allowing it to recover the accuracy of a full-tree implementation.

It may be possible to tune the placement of the approximation terms to be better suited for the larger range present in anomaly detection, but this would result in worse performance for the other benchmarks. In the pursuit of a generally applicable delay space systolic array we optimize for the common case. However, this does mean some applications and networks may not be suitable for this approach. This histogram analysis can be easily used to determine if a new network will perform well on a given accelerator architecture.

Arch.	Image Recognition	Mobile-net	Keyword Spotting	Anomaly Detection
DS DM	1.04	2.33	0.49	0.16
DS TM Tiny	1.87	3.32	0.68	0.34
SATA	5.14	8.5	3.5	0.66
Digital	0.26	0.45	0.09	0.05

Table 3: Inference latency (in *ms*) of the delay space systolic array compared to the SATA SNN architecture and a digital systolic array.

6.2 Architectural Analysis

Using the evaluation methodology discussed in Section 5 we perform a design space exploration for two of the MLPerf Tiny benchmarks for both digital and temporal memory systems, shown in Figure 11. Our design space exploration varies the height and width of the systolic array, along with the size of each processing element. We fix the input buffer, output buffer and filter map to 32KB, and use 2×12 and 25 approximation terms for nLSE and nLDE respectively with a 1ns temporal mapping. Each network is quantized to 8 bits and use a WS dataflow. We show the area of the systolic array on the y-axis, while the x-axis shows the energy delay product (EDP) of the systolic array and on-chip memory, excluding leakage power. For this design space exploration we only consider architectures that have a smaller physical footprint than an 8 bit 12×14 digital systolic array, showing how a single delay space MAC unit is significantly smaller than a digital multiplier.

While the fundamental MAC unit is small, the nLDE and TDC are quite large, limiting the maximum possible width for designs that meet the area constraint when using the digital memory system. Because of this limit, the exploration reveals that for edge inference tasks, narrow (smaller broadcasts) and tall (more IF map bandwidth) designs tend to provide better EDP with a smaller footprint, with most Pareto optimal designs having a width between 7 and 9 PEs. Conversely, the temporal memory system can have wider arrays because the expensive TDCs are no longer necessary, creating many 2-tree designs along the Pareto optimal frontier. Additionally, the visual wake word application benefits greatly from 2-tree designs, while for all other benchmarks and designs the 4-tree designs perform better.

Using this analysis we select 3 designs that are generally along the Pareto optimal frontier. We select a 9×8 array with a 4-tree PE from both designs to highlight the benefits of temporal memory, referred to as DS DM (digital memory) and DS TM (temporal memory) for comparison against other systolic arrays. We also select a temporal memory design that is less than half the size of a digital array as another comparison point, referred to as DS TM Tiny.

We then compare these designs to an 8-bit 12×14 digital systolic array, a common edge inference configuration [12], and SATA, a SNN accelerator. We evaluate SATA using SATA-sim, and assume each benchmark is implemented with 20 time steps, a conservative estimate as most direct ANN-to-SNN networks use 32 to 1000 time steps [25]. For both designs we assume the same DRAM configuration as our design space exploration, then show the on-chip inference energy consumption of the architectures for each benchmark in Figure 12. We show the latency for a single inference in

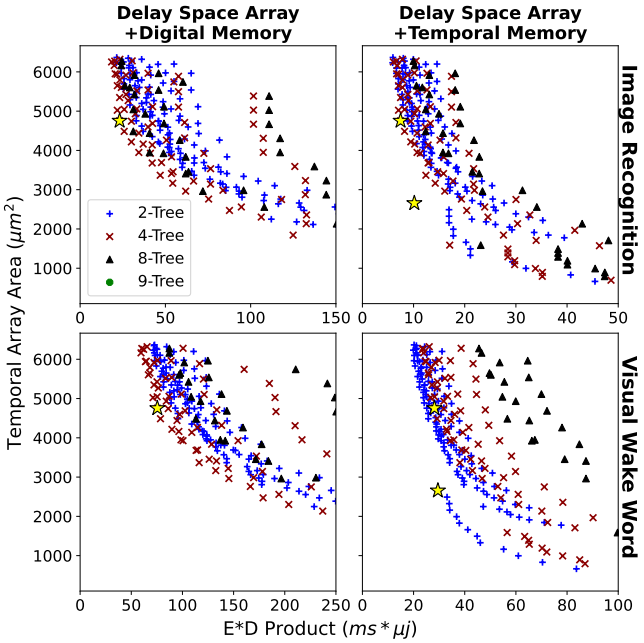


Figure 11: Design space exploration of potential temporal array sizes, comparing the overall area to the EDP. 3 designs are highlighted across the benchmarks with a star corresponding to DS DM, DS TM and DS TM Tiny.

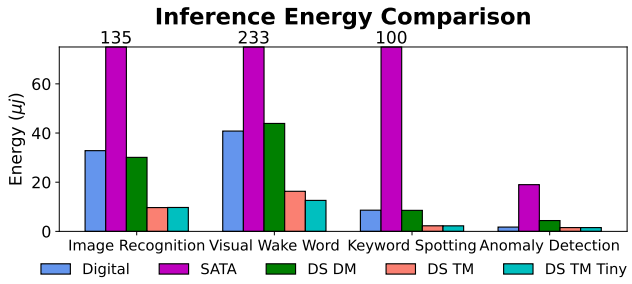


Figure 12: Active Energy comparison between our proposed delay space systolic array, an 8-bit digital implementation and an SNN systolic array, SATA [66].

Table 3, excluding DS TM since it has roughly the same performance as DS DM.

Compared to SATA, our hybrid systolic array performs much better, even when using digital memories. DS DM achieves an 11× improvement in energy efficiency over SATA for keyword spotting, while performing 4.5× better for image recognition. This also comes with significant speedups over the SNN approach, saving over 0.5 to 6ms across the benchmark tasks. These results show the currently attainable benefits of our unified spiking ANN approach, leveraging classical dataflows for significant energy efficiency over methods leveraging temporal dynamics. While it has been shown that SNNs can leverage fewer time steps when utilizing advanced retraining schemes, DS DM still provides at least 2× energy improvement for each benchmark over a 6-step network on SATA while remaining strictly faster. However, when compared to a digital accelerator this digital memory approach consumes roughly the same amount of energy as a digital array, with slight variations across the benchmarks. While in practice digital arrays need to slow significantly to achieve this energy efficiency [1], the EDP difference is significant.

This stems in part due to the cost of extra domain conversions, and in part due to the memory systems dominating the cost of inference. This illustrates the need for temporal memory in order to drive continued improvements in energy efficiency. Using this design our systolic array becomes significantly more energy efficient, even when using an incredibly small design. DS TM and DS TM Tiny provide 2.5× to 3.5× improvement in energy consumption compared to the digital design for all benchmarks except for anomaly detection, while achieving 12× to 45× better energy efficiency compared to SATA. This shows the potential future for our proposed array, demonstrating how operations within a fully temporal system can provide significantly better energy efficiency than even the most aggressive digital approaches.

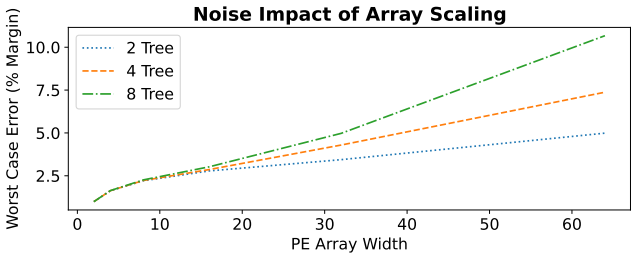


Figure 13: Analysis of broadcast noise as the width of the array scales.

6.3 Implications of Broadcast Scaling

While we show several design points that provide very energy efficient inference for the MLPerf Tiny benchmarks while maintaining accuracy, the final question is how this technique scales to larger systolic arrays. In terms of accuracy, larger networks tend to be more resilient to error, especially when trained with regularization techniques. This can be seen in Figure 8 and Table 2, where the visual wake word application is much less affected by both the number of terms and the PE tree size. This application leverages the Mobilenet [24] network architecture, which is by far the largest network of the benchmark suite, showing how size can lead to better noise resilience.

For larger networks, a larger array may be necessary to meet performance goals. This can generally be achieved in two ways: increasing the size of the array (scale up), or adding more arrays in a tile-wise manner (scale out). The trade-offs between the two have been well studied [46], and the delay space systolic array shares many of the same trade-offs because it leverages the same dataflows as ANNs. The only additional complexity is the broadcast operation, which introduces increasingly more noise as the array grows.

We analyze this impact in Figure 13, where we assume a system that augments the broadcast operation with buffers at every split of the broadcast tree, and a buffer every 50μm to ensure sharp

rising/falling edges. We calculate the maximum error from this system assuming a maximum voltage variation of 5 mV in a system with a 700 mV supply, and three standard deviations of RJ. This maximum timing error is then divided by the timing margin that would cause a TDC at the PE to flip a bit.

Larger PE trees introduce noise at a faster rate as the array width scales because each PE has a physically larger footprint. This causes the required broadcast distance to increase, allowing for larger worst case error. However, even the larger designs stay well under the timing margin, even assuming worst case noise. This small amount of noise, coupled with noise-robust neural networks, allows the array to scale to several times the size of the arrays we've analyzed.

7 Related Work

There has been a large body of work in digital systolic arrays that focuses on minimizing memory accesses and data movement [11, 12, 65]. While this clearly represents an important area of research, the digital MAC operation establishes a lower energy bound, and alternative arithmetic options must be investigated to push for low energy operations.

Spiking neural networks[21] have attempted to resolve this issue by changing the base neuron model to a leaky-integrate-and-fire model [54], which removes the need for expensive multiplication operations. Several industrial chips have been developed, such as Loihi [17] and TrueNorth [5], but it has proven difficult to achieve similar accuracy to ANNs while still maintaining the energy efficiency advantage [20].

While others have explored the connection between ANNs and SNNs in the past, with prior schemes using unique neurons, such as leaky-integrate-and-fire or integrate-and-fire, these cannot be mapped exactly to the functions used in ANNs. This imperfect mapping introduces several types of errors that are difficult to overcome [58]. Additionally, these conversions are implemented traditionally with digital architectures, and require either a large number of timesteps [31] or multi-valued spikes to recoup the accuracy of ANN networks. SNNs that require a large number of timesteps have been shown to have very poor latency and energy efficiency [50] while multi-valued spikes allow for fewer time steps but then require matrix multiplication on a similar scale to ANNs.

Alternatively, several unary computing approaches have been investigated in recent years, particularly focused on stochastic computing accelerators [32]. This approach uses a random bitstream to perform energy efficient computation. Some approaches leverage emerging technologies to generate the random values [16], while others rely on expensive digital random number generators [61]. This approach allows for incredibly energy efficient computation, but requires a large number of cycles for values to stabilize. The random number generation either consumes significant hardware resources or is difficult to implement. Finally, creating a configurable architecture requires a large number of conversions between domains, resulting in more expensive inference costs [62].

Other work has instead focused on purely temporal computation, using wavefront computing to perform energy efficient DNA sequencing [14, 15, 36] and graph operations [35]. Boosted race trees [56] even leverage the basic race logic framework for energy efficient decision tree inference. However, race trees have been shown to have limited effectiveness when applied to larger inference tasks [56]. Compac [49] uses an analog temporal memory to perform addition and multiplication, thus allowing it to perform inference. While this achieves energy efficiency it comes at the cost of performance since the operations become linear, forcing iterative multiplications.

8 Conclusion

We present a new approach to machine learning inference which bridges the gap between spiking neural networks (SNNs) and traditional artificial neural networks (ANNs) through temporal arithmetic. We show how temporal computation can be extended to implement a log-transformed version of neural networks by interpreting neural weights as delay elements and activations as a single "spike" in time. This transformation simplifies the dot-product calculation at the core of the network, and uses carefully optimized approximations for the simplified computations.

Implementing full neural networks directly in this log-time domain introduces significant new challenges and opportunities. First, network weights must be easily loaded and unloaded while also influencing the delay of signals in the system to achieve the required scaling. We develop a hybrid temporal/digital approach that enables weights to be migrated fully digitally while allowing delay operations to remain entirely in the temporal domain. Further improvements can be achieved by integrating previously demonstrated temporal memory systems — such systems have never been evaluated in the context of an end-to-end application. Additionally, we show that temporal arithmetic presents unexpected opportunities for energy savings in neural network systems. Two of the most common additional operators, max-pooling and ReLU, are not only exceedingly low-cost (e.g., max can be implemented with a single logic gate) but also simplify temporal arithmetic by eliminating much of its complexity — for example, ReLU removes the need for negative rails and their interactions with both positive and negative rails.

Through a careful analysis of these systems from low-level circuit noise up to application-level evaluation, we show how such a scheme is not only viable, but that the different hardware parameters can impact the accuracy of the converted neural network, creating a new set of energy and accuracy trade offs. By effectively marrying the simplicity and energy efficiency of SNNs with the well-established structure and functionality of ANNs, this work opens new avenues for research and development in low power embedded machine learning systems.

References

[1] 2024. https://mlcommons.org/benchmarks/inference-tiny/
[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.
[3] Abien Fred Agarap. 2018. Deep learning using rectified linear units (ReLU). *arXiv preprint arXiv:1803.08375* (2018).

[4] Ankur Agrawal, Sae Kyu Lee, Joel Silberman, Matthew M. Ziegler, Mingu Kang, Swagath Venkataramani, Nianzheng Cao, Bruce M. Fleischer, Michael Guillorn, Matthew Cohen, Silvia Melitta Mueller, Jinwook Oh, Martin Lutz, Jinwook Jung, Siyuranga O. Koswatta, Ching Zhou, Vidhi Zalani, James Bonanno, Robert Casatuta, Chia-Yu Chen, Jungwook Choi, Howard Haynie, A. Herbert, Radhika Jain, Monodeep Kar, Kyu-Hyoun Kim, Yulong Li, Zhibin Ren, Scot Rider, Marcel Schaal, Kerstin Schelm, Michael Scheuermann, Xiao Sun, Hung Tran, Naigang Wang, Wei Wang, Xin Zhang, Vinay Shah, Brian W. Curran, Vijayalakshmi Srinivasan, Pong-Fei Lu, Sunil Shukla, Leland Chang, and K. Gopalakrishnan. 2021. A 7nm 4-Core AI Chip with 25.6TFLOPS Hybrid FP8 Training, 102.4TOPS INT4 Inference and Workload-Aware Throttling. *2021 IEEE International Solid- State Circuits Conference (ISSCC)* 64 (2021), 144–146. https://api.semanticscholar.org/CorpusID:232152824

[5] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. 2015. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE transactions on computer-aided design of integrated circuits and systems* 34, 10 (2015), 1537–1557.

[6] Kartik Audhkhasi, Osonde Osoba, and Bart Kosko. 2016. Noise-enhanced convolutional neural networks. *Neural Networks* 78 (2016), 15–23.

[7] Rajeev Balasubramonian, Andrew B Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New tools for interconnect exploration in innovative off-chip memories. *ACM Transactions on Architecture and Code Optimization (TACO)* 14, 2 (2017), 1–25.

[8] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, Urmish Thakker, Antonio Torrini, Peter Warden, Jay Cordaro, Giuseppe Di Guglielmo, Javier Duarte, Stephen Gibellini, Videet Parekh, Honson Tran, Nhan Tran, Niu Wenxu, and Xu Xuesong. 2021. MLPerf Tiny Benchmark. arXiv:2106.07597

[9] Romain Brette. 2015. Philosophy of the spike: rate-based vs. spike-based theories of the brain. *Frontiers in systems neuroscience* 9 (2015), 151.

[10] Qiankai Cao, Xi Chen, and Jie Gu. 2023. Development of Tropical Algebraic Accelerator with Energy Efficient Time-Domain Computing for Combinatorial Optimization and Machine Learning. In *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 1–6.

[11] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. 2014. DaDianNao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 609–622.

[12] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits* 52, 1 (2016), 127–138.

[13] Zhengyu Chen and Jie Gu. 2016. Analysis and design of energy efficient time domain signal processing. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. 100–105.

[14] Zhengyu Chen and Jie Gu. 2019. 19.7 A scalable pipelined time-domain DTW engine for time-series classification using multibit time flip-flops with 140Giga-cell-updates/s throughput. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 324–326.

[15] Zhengyu Chen and Jie Gu. 2020. High-throughput dynamic time warping accelerator for time-series classification with pipelined mixed-signal time-domain computing. *IEEE Journal of Solid-State Circuits* 56, 2 (2020), 624–635.

[16] Matthew W Daniels, Advait Madhavan, Philippe Talatchian, Alice Mizrahi, and Mark D Stiles. 2020. Energy-efficient stochastic computing with superparamagnetic tunnel junctions. *Physical review applied* 13, 3 (2020), 034016.

[17] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro* 38, 1 (2018), 82–99.

[18] Asma Dehghani, Mohsen Saneei, and Ali Mahani. 2016. A high-resolution time-to-digital converter using a three-level resolution. *International Journal of Electronics* 103, 8 (2016), 1248–1261.

[19] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P Jouppi. 2012. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 7 (2012), 994–1007.

[20] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. 2015. Backpropagation for energy-efficient neuromorphic computing. *Advances in neural information processing systems* 28 (2015).

[21] Samanwoy Ghosh-Dastidar and Hojjat Adeli. 2009. Spiking neural networks. *International journal of neural systems* 19, 04 (2009), 295–308.

[22] Rui Graça, Brian McReynolds, and Tobi Delbruck. 2023. Shining light on the DVS pixel: A tutorial and discussion about biasing and optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4045–4053.

[23] Rhys Gretsch, Peiyang Song, Advait Madhavan, Jeremy Lau, and Timothy Sherwood. 2024. Energy Efficient Convolutions with Temporal Arithmetic. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 354–368.

[24] Andrew G Howard. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[25] Yangfan Hu, Qian Zheng, Guoqi Li, Huajin Tang, and Gang Pan. 2024. Toward Large-scale Spiking Neural Networks: A Comprehensive Survey and Future Directions. *arXiv preprint arXiv:2409.02111* (2024).

[26] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2704–2713.

[27] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.

[28] Edin Kadric, Paul Gurniak, and André DeHon. 2016. Accurate parallel floating-point accumulation. *IEEE Trans. Comput.* 65, 11 (2016), 3224–3238.

[29] Anders Krogh and John Hertz. 1991. A simple weight decay can improve generalization. *Advances in neural information processing systems* 4 (1991).

[30] Can Li, Miao Hu, Yunning Li, Hao Jiang, Ning Ge, Eric Montgomery, Jiaming Zhang, Wenhao Song, Noraica Dávila, Catherine E Graves, et al. 2018. Analogue signal and image processing with large memristor crossbars. *Nature electronics* 1, 1 (2018), 52–59.

[31] Yang Li, Dongcheng Zhao, and Yi Zeng. 2022. Bsnn: Towards faster and better conversion of artificial neural networks to spiking neural networks with bistable neurons. *Frontiers in neuroscience* 16 (2022), 991851.

[32] Yidong Liu, Siting Liu, Yanzhi Wang, Fabrizio Lombardi, and Jie Han. 2020. A survey of stochastic computing neural networks for machine learning applications. *IEEE Transactions on Neural Networks and Learning Systems* 32, 7 (2020), 2809–2824.

[33] Sangkug Lym and Mattan Erez. 2020. FlexSA: Flexible systolic array architecture for efficient pruned DNN model training. *arXiv preprint arXiv:2004.13027* (2020).

[34] Advait Madhavan, Matthew W Daniels, and Mark D Stiles. 2021. Temporal state machines: Using temporal memory to stitch time-based graph computations. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 17, 3 (2021), 1–27.

[35] Advait Madhavan, Timothy Sherwood, and Dmitri Strukov. 2014. Race logic: A hardware acceleration for dynamic programming algorithms. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 517–528.

[36] Advait Madhavan, Timothy Sherwood, and Dmitri Strukov. 2017. A 4-mm 2 180-nm-CMOS 15-giga-cell-updates-per-second DNA sequence alignment engine based on asynchronous race conditions. In *2017 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 1–4.

[37] Advait Madhavan and Mark D Stiles. 2020. Storing and retrieving wavefronts with resistive temporal memory. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[38] Agnish Mal, Amit Krishna Dwivedi, and Aminul Islam. 2016. A comparative analysis of various programmable delay elements using predictive technology model. In *2016 International Conference on Microelectronics, Computing and Communications (MicroCom)*. 1–5. https://doi.org/10.1109/MicroCom.2016.7522456

[39] M. Maymandi-Nejad and M. Sachdev. 2003. A digitally programmable delay element: design and analysis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 11, 5 (2003), 871–878. https://doi.org/10.1109/TVLSI.2003.810787

[40] Daisuke Miyashita, Edward H Lee, and Boris Murmann. 2016. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025* (2016).

[41] Xunjun Mo, Jiaqi Wu, Nijwm Wary, and Tony Chan Carusone. 2021. Design Methodologies for Low-Jitter CMOS Clock Distribution. *IEEE Open Journal of the Solid-State Circuits Society* 1 (2021), 94–103. https://api.semanticscholar.org/CorpusID:238750409

[42] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. 2021. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295* (2021).

[43] Jens E. Pedersen, Steven Abreu, Matthias Jobst, Gregor Lenz, Vittorio Fra, Felix Christian Bauer, Dylan Richard Muir, Peng Zhou, Bernhard Vogginger, Kade Heckel, Gianvito Urgese, Sadasivan Shankar, Terrence C. Stewart, Sadique Sheik, and Jason K. Eshraghian. 2024. Neuromorphic intermediate representation: A unified instruction set for interoperable brain-inspired computing. *Nature Communications* 15, 1 (Sept. 2024), 8122. https://doi.org/10.1038/s41467-024-52259-9

[44] Juan Riquelme and Ioannis Vourkas. 2024. A Star Network of Bipolar Memristive Devices Enables Sensing and Temporal Computing. *Sensors* 24, 2 (2024), 512.

[45] Mohammad Nazmus Sakib, Rahul Sreekumar, Vaibhav Verma, Tommy Tracy, and Mircea R Stan. 2021. Atcpim: Analog to time coded processing in memory for iot at the edge. In *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*. IEEE, 704–709.

[46] Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2020. A systematic methodology for characterizing scalability of DNN accelerators using SCALE-Sim. In *2020 IEEE International*

Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE, 58–68.

[47] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. SCALE-Sim: Systolic CNN accelerator simulator. *arXiv preprint arXiv:1811.02883* (2018).

[48] Satyabrata Sarangi and Bevan Baas. 2021. DeepScaleTool: A Tool for the Accurate Estimation of Technology Scaling in the Deep-Submicron Era. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. https://doi.org/10.1109/ISCAS51556.2021.9401196

[49] Aseem Sayal, Shirin Fathima, SS Teja Nibhanupudi, and Jaydeep P Kulkarni. 2020. Compac: Compressed time-domain, pooling-aware convolution CNN engine with reduced data movement for energy-efficient AI computing. *IEEE Journal of Solid-State Circuits* 56, 7 (2020), 2205–2220.

[50] Guobin Shen, Dongcheng Zhao, Tenglong Li, Jindong Li, and Yi Zeng. 2024. Are Conventional SNNs Really Efficient? A Perspective from Network Quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 27538–27547.

[51] James Smith. 2018. Space-Time Algebra: A Model for Neocortical Computation. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 289–300. https://doi.org/10.1109/ISCA.2018.00033

[52] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1 (2014), 1929–1958.

[53] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. 2008. The missing memristor found. *nature* 453, 7191 (2008), 80–83.

[54] Corinne Teeter, Ramakrishnan Iyer, Vilas Menon, Nathan Gouwens, David Feng, Jim Berg, Aaron Szafer, Nicholas Cain, Hongkui Zeng, Michael Hawrylycz, et al. 2018. Generalized leaky integrate-and-fire models classify multiple neuron types. *Nature communications* 9, 1 (2018), 709.

[55] Albert Theuwissen. 2021. There's More to the Picture Than Meets the Eye, and in the future it will only become more so. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. 30–35. https://doi.org/10.1109/ISSCC42613.2021.9366058

[56] Georgios Tzimpragos, Advait Madhavan, Dilip Vasudevan, Dmitri Strukov, and Timothy Sherwood. 2019. Boosted race trees for low energy classification. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 215–228.

[57] Hamed Vakili, Mohammad Nazmus Sakib, Samiran Ganguly, Mircea Stan, Matthew W Daniels, Advait Madhavan, Mark D Stiles, and Avik W Ghosh. 2020. Temporal memory with magnetic racetracks. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 6, 2 (2020), 107–115.

[58] Yuchen Wang, Hanwen Liu, Malu Zhang, Xiaoling Luo, and Hong Qu. 2024. A universal ANN-to-SNN framework for achieving high accuracy and low latency deep Spiking Neural Networks. *Neural Networks* 174 (2024), 106244.

[59] Refael Whyte, Lee Streeter, Michael J Cree, and Adrian A Dorrington. 2015. Application of lidar techniques to time-of-flight range imaging. *Applied optics* 54, 33 (2015), 9654–9664.

[60] William Wilson, Tom Chen, and Ryan Selby. 2013. A current-starved inverter-based differential amplifier design for ultra-low power applications. In *2013 IEEE 4th Latin American symposium on circuits and systems (LASCAS)*. IEEE, 1–4.

[61] Di Wu, Jingjie Li, Ruokai Yin, Hsuan Hsiao, Younghyun Kim, and Joshua San Miguel. 2020. UGEMM: Unary computing architecture for GEMM applications. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 377–390.

[62] Di Wu and Joshua San Miguel. 2022. uSystolic: Byte-crawling unary systolic array. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 12–24.

[63] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. 2020. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602* (2020).

[64] Thucydides Xanthopoulos. 2009. *Clocking in modern VLSI systems*. Springer Science & Business Media.

[65] Lei Yang, Zheyu Yan, Meng Li, Hyoukjun Kwon, Liangzhen Lai, Tushar Krishna, Vikas Chandra, Weiwen Jiang, and Yiyu Shi. 2020. Co-Exploration of Neural Architectures and Heterogeneous ASIC Accelerator Designs Targeting Multiple Tasks. arXiv:2002.04116 [cs.LG]

[66] Ruokai Yin, Abhishek Moitra, Abhiroop Bhattacharjee, Youngeun Kim, and Priyadarshini Panda. 2023. SATA: Sparsity-Aware Training Accelerator for Spiking Neural Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 6 (2023), 1926–1938. https://doi.org/10.1109/TCAD.2022.3213211

[67] Qiaochu Zhang, Shiyu Su, and Mike Shuo-Wei Chen. 2022. A cost-efficient fully synthesizable stochastic time-to-digital converter design based on integral nonlinearity scrambling. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 1021–1026.

[68] Jiawei Zhao, Steve Dai, Rangharajan Venkatesan, Brian Zimmer, Mustafa Ali, Ming-Yu Liu, Brucek Khailany, William J Dally, and Anima Anandkumar. 2022. Lns-madam: Low-precision training in logarithmic number system using multiplicative weight update. *IEEE Trans. Comput.* 71, 12 (2022), 3179–3190.

[69] Wei Zhao and Yu Cao. 2006. New generation of predictive technology model for sub-45 nm early design exploration. *IEEE Transactions on electron Devices* 53, 11 (2006), 2816–2823.