Energy Efficient Convolutions with Temporal Arithmetic

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

Peiyang Song
p_song@ucsb.edu
UC Santa Barbara, USA

Advait Madhavan
amadha1@umd.edu
University of Maryland, USA

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

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

Abstract

Convolution is an important operation at the heart of many applications, including image processing, object detection, and neural networks. While data movement and coordination operations continue to be important areas for optimization in general-purpose architectures, for computation fused with sensor operation, the underlying multiply-accumulate (MAC) operations dominate power consumption. Non-traditional data encoding has been shown to reduce the energy consumption of this arithmetic, with options including everything from reduced-precision floating point to fully stochastic operation, but all of these approaches start with the assumption that a complete analog-to-digital conversion (ADC) has already been done for each pixel. While analog-to-time converters have been shown to use less energy, arithmetically manipulating temporally encoded signals beyond simple min, max, and delay operations has not previously been possible, meaning operations such as convolution have been out of reach. In this paper we show that arithmetic manipulation of temporally encoded signals is possible, practical to implement, and extremely energy efficient.

The core of this new approach is a negative log transformation of the traditional numeric space into a 'delay space' where scaling (multiplication) becomes delay (addition in time). The challenge lies in dealing with addition and subtraction. We show these operations can also be done directly in this negative log delay space, that the associative and commutative properties still apply to the transformed operations, and that accurate approximations can be built efficiently in hardware using delay elements and basic CMOS logic elements. Furthermore, we show that these operations can be chained together in space or operated recurrently in time. This approach fits naturally into the staged ADC readout inherent to most modern cameras. To evaluate our approach, we develop a software system that automatically transforms traditional convolutions into delay space architectures. The resulting system is used to analyze and balance error from both a new temporal equivalent of quantization and delay element noise, resulting in designs that improve the energy per pixel of each convolution frame by more than 2× compared to a state-of-the-art while improving the energy delay product by four orders of magnitude.

CCS Concepts: • **Computer systems organization** → **Embedded hardware**; • **Hardware** → **Emerging technologies**.

ACM Reference Format:
Rhys Gretsch, Peiyang Song, Advait Madhavan, Jeremy Lau, and Timothy Sherwood. 2024. Energy Efficient Convolutions with Temporal Arithmetic. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3620665.3640395

ASPLOS '24, April 27-May 1, 2024, La Jolla, CA, USA

ACM ISBN 979-8-4007-0385-0/24/04.
https://doi.org/10.1145/3620665.3640395

1 Introduction

Convolution is a critical operation for image processing, being the basis for feature extraction [31], filters [38], and edge detection [50]. The rise of visual sensor networks [10] and convolutional neural networks (CNNs) [24] have further amplified the importance of convolutions. In the domain of sensor-embedded computation specifically, where convolution operations are typically local and resident, energy consumption is dominated by the cost of converting the data into a form that can be manipulated digitally and the cost of multiply and accumulate (MAC) operations [42].

Technology scaling helps reduce energy consumption but the slowing of traditional computational scaling, coupled with ever increasing sensor array densities, creates opportunities for non-traditional computing paradigms. While this can be as simple as reduced-precision floating point [20], more radical approaches include stochastic computing [47, 48] and race logic [27], which experiment with alternative data encodings. These data encodings are of particular interest because they leverage the electrical behavior of basic

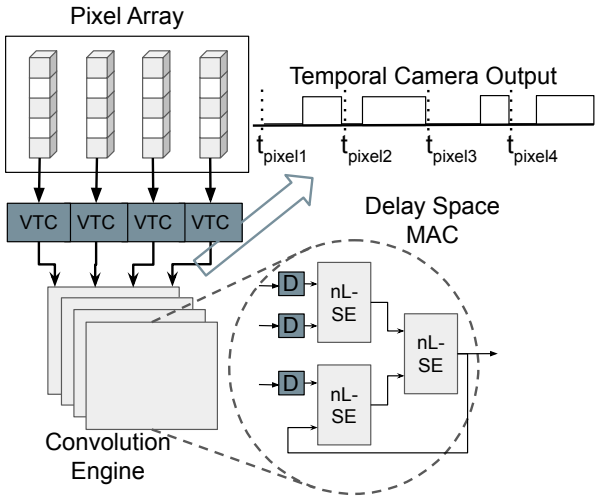


Figure 1. A full system diagram of our architecture. The ADCs in the staged pixel readout are replaced with voltage to time (VTC). An example of the output from one of these VTCs is shown above the system. This output is then passed to the hard-coded convolution engine. The core of this architecture is our delay space MAC where the output can be routed back to the input, creating our recurrence architecture.

binary logic gates in new ways. Race logic encodes information into the timing of a voltage edge and relies on four operations: minimum, maximum, (with AND and OR respectively), delay, and inhibit [27, 43] to perform computation. These operations are logically complete [41] and have a near minimal activity factor [28], creating the potential for incredibly energy efficient computation. However, to the best of our knowledge no prior work has demonstrated efficient arithmetic operations with race logic primitives, limiting the general applicability of race logic.

A primary contribution of this work is a new temporal value encoding — one that fully leverages the existing race-logic circuits, but that also allows for efficient arithmetic — while still representing values as a single edge.

The key insight is to apply a mathematical transformation under which addition and multiplication are co-transformed with the encoded values, forming a new mathematical ring over delays. Data values in traditional *importance space* are converted into a *delay space* with a rising edge occurring after a *delay equal to the negative log of the value.* Multiplication in importance space becomes simple addition in delay space, which means that multiplication can be implemented with a simple delay operation. Addition, however, becomes a negative log sum exponential (nLSE) function. While the nLSE function initially seems challenging to implement, we show that the function can be efficiently approximated with nothing more than min, max, and delay.

While delay space is interesting theoretically and opens many new avenues for temporal computation, it is also practical to implement, tolerant to noise, and highly energy efficient in practice. To demonstrate these points, we design and evaluate a near-sensor architecture operating completely in delay space, from sensor activation through convolution, as shown in Figure 1.

This proposed architecture matches the iterative row-by-row read-out of most sensor arrays with a temporal scaling and summation that can be applied iteratively as each new row is read. At the core of this convolution system is an automated transformation that converts the MAC operations into nLSE and delay units in a novel recurrence architecture. This work presents the following contributions:

- A new data encoding that transforms the linear operations of convolution into new operations in an inverted logarithmic delay space, significantly expanding the computational potential of temporal computing.
- Efficient approximations for the new "soft" operations required in delay space, using existing "hard" temporal logic primitives, and recurrent hardware implementations of those approximations to iteratively perform delay-coded summations.
- An architectural evaluation tool for our convolution architecture and the hardware implementations of these temporal computations. We demonstrate the utility of this approach by embedding these linear operations within the natural temporal staging of pixel read-out.
- We quantify the impact of various noise factors on approximation accuracy and show that realistic designs can operate with energy efficiencies 8x better than prior work in the area.

We start with description of the encoding, its properties, and hardware implementation (Sections 2 and 3) before moving on to a more complete architecture and evaluation (Sections 4 and 5), finishing with connections to other work and final conclusions (Sections 6 and 7).

2 Delay Space Arithmetic

In race logic, a signal's time of arrival – *when* a rising or falling edge occurs – encodes the signal's value, rather than the signal's voltage level. Using just four basic operations on those signals, first arrival (FA), last arrival (LA), INHIBIT, and DELAY [27, 43], arbitrary temporal functions can be constructed [41]. If one thinks about the delay in time as linearly encoding a value (as has been assumed by prior work), then FA and LA execute min and max functions respectively, while the delay performs simple addition. This makes some intuitive sense as a signal computed as FA(τ_1,DELAY(τ_2, δ)) will appear at time $\min(\tau_1, \tau_2 + \delta)$.

This encoding has the advantage of a very simple hardware mapping, with FA and LA on rising edges being implemented by simple OR and AND gates respectively. While

not corresponding exactly to existing logic gates, INHIBIT only requires two transistors [43], and there are a myriad of hardware delay elements [30].

In addition to being a complete temporal logic, race logic has been shown to efficiently implement shortest path graph algorithms, decision trees, sorting networks, and other useful constructions [27, 41, 43]. However, efficient implementations of more general arithmetic operations with race logic primitives has never been demonstrated, in part due to the complexity of multiplication and the inability to represent negative numbers or perform subtraction. Ideally we would be able to achieve the following properties:

1. **Operations directly on encoded form**: Given a real number we need a clear way to get into and out of this encoding coupled with a way to perform the operations necessary for convolution (addition, subtraction, and multiplication) directly in the encoded form, *without* unnecessary conversions. More formally we could say we are looking for a bijective ring homomorphism of the reals.
2. **Important values early**: Traditionally, a signal of larger magnitude is represented with a larger number. When dealing with delays, it would be far more natural for highly important values to be encoded as *shorter* durations of time, so that less important contributions can be truncated at any time. The more "excited" the system, the smaller the delay and hence the larger the importance – meaning the events that occur first are those that carry the most weight.
3. **Broader dynamic range**: From image processing to machine learning, the ability to stretch beyond the constraints of linear encoding is very useful. This is particularly critical when dealing with delays because the execution time and energy consumption is coupled to the data representation. Most prior "unary" schemes map values linearly, which makes it difficult to deal with very large and very small values in the same computation. An ideal solution would allow a broad range of values to be operated on in a way similar in spirit to a floating point representations, without introducing the problems of normalization.

We propose that a new encoding meeting all of these requirements is a negative log mapping, where a value x in the original convolution maps to a signal that has a rising edge after a time delay of x' where

$$x' = -\ln(x) \tag{1}$$

Values encoded in this way are in "delay space", and we use the notation of primes such as x' to indicate delay space values. Given a delay x' we can map its value back to the original "importance space" via

$$x = \frac{1}{e^{x'}} = e^{-x'} \tag{2}$$

All operations in importance space convolution have direct equivalents in delay space, which can be derived by applying Equation 1 and logarithmic identities. A delay space multiplication can be performed using simple addition. Delay space addition and subtraction map to the negative log sum exponential (nLSE) and negative log difference exponential (nLDE) functions respectively:

$$x \cdot y \mapsto x' + y' \tag{3}$$

$$x + y \mapsto -\ln(e^{-x'} + e^{-y'}) = \text{nLSE}(x', y') \tag{4}$$

$$x - y \mapsto -\ln(e^{-x'} - e^{-y'}) = \text{nLDE}(x', y') \tag{5}$$

2.1 Approximate Delay Space Addition

At first glance this negative log space seems awkward – the nLSE and nLDE functions in particular seem difficult to efficiently implement in hardware. But delay space has some nice properties which we can exploit. First, as values approach infinity in importance space, they approach zero time in delay space, so more important values have less delay.

Second, because it is a logarithmic encoding, the value space gives reasonable encodings to a far wider range of values. Furthermore we can *re-scale* our values by simply shifting the reference point for the delay, because addition or subtraction in delay space is the same as multiplicative scaling in importance space. This means we have a much improved range to operate in. Finally, and most critically from an implementation standpoint, the nLSE function is a form of "soft min" operation (where LSE is the "real soft max" used commonly in machine learning [3]) and is associative, commutative, and addition distributes through it as $\text{nLSE}(a+\delta, b+\delta) = \text{nLSE}(a,b) + \delta$. These properties, along with its bounds, allow us to come up with arbitrarily tight approximations for nLSE using only min, max, and delay.

Figure 2 shows the nLSE function (Equation 4). The function is bounded from above by $\min(x', y')$ and at the most extreme points, where one of x' or y' is much larger than the other, the behavior of $\text{nLSE}(x', y')$ converges to $\min(x', y')$. However, as x' and y' become closer in value, nLSE deviates further and further from min, with worst-case error $-\ln(2)$ when $x' = y'$.

While the "hard min" of $\min(x', y')$ can serve as an approximation of nLSE, introducing a shifted max-term ($\max(x' + C, y' + D)$) under the min adds a valley to the approximation, which can reduce approximation error. An arbitrary number of these max-terms can be added, and approximation error can be made arbitrarily small with more max-terms. An approximation with n max-terms has the form:

$$\begin{aligned} &\min(x', y', \max(x' + C_0, y' + D_0), \\ &\qquad \dots \\ &\qquad \max(x' + C_{n-1}, y' + D_{n-1})) \end{aligned} \tag{6}$$

To improve our nLSE approximation with max-terms, we must find appropriate values for the constants C_i and D_i.

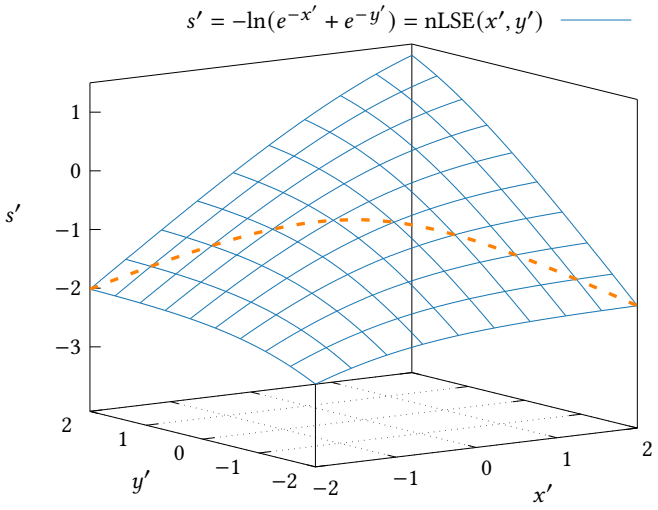


Figure 2. Plot of $s' = \mathrm{nLSE}(x', y')$. The surface has the same shape along all planes where $x' + y' = K$, for all constants K. The dashed line shows a representative slice of the surface on the plane where $x' + y' = 0$. This representative slice is shown again in Figure 3.

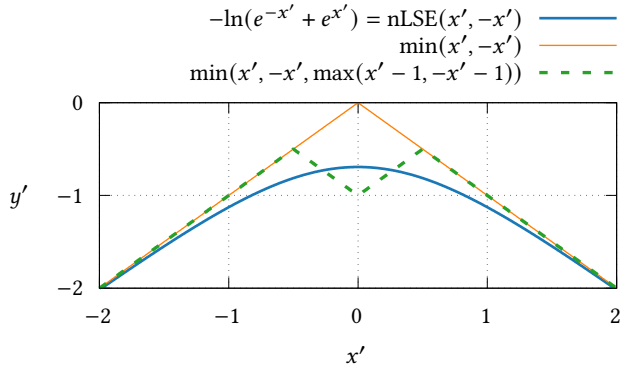


Figure 3. Plot of $\mathrm{nLSE}(x', -x')$. This is the representative slice of nLSE from Figure 2, where $x' + y' = 0$. This graph also shows how min roughly approximates nLSE, and how introducing a max-term can improve the approximation.

To do so, we first observe that the curves of $\mathrm{nLSE}(x', y')$ in Figure 2 all have the same shape for any arbitrary constant $K = x' + y'$. The dotted line in Figure 2 shows a representative slice of this surface, where $x' + y' = 0$. All slices parallel to this representative slice have the same shape. Therefore, without loss of generality, we can set $K = 0$ and focus on approximating $\mathrm{nLSE}(x', -x')$. This observation allows us to simplify the problem from two inputs x', y' to one input x'. Figure 3 shows nLSE simplified to one input, how min roughly approximates nLSE, and an improved approximation with one max-term, where $C_0 = D_0 = -1$.

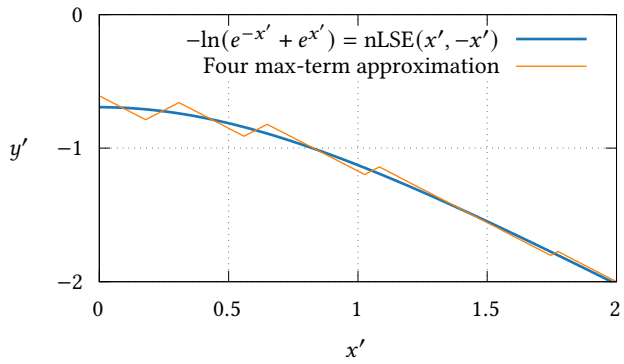


Figure 4. Approximating nLSE with four max-terms. We only approximate positive x' values because Figure 3 is symmetric about the y'-axis.

Figure 3 reveals that this simplified one-input nLSE is symmetric over the y'-axis, so it is sufficient to approximate only non-negative x' values. If negative x' values are involved, we can negate the inputs and thus simplify the calculation down to non-negative nLSE. This same trick works for two-input nLSE, which can be computed after swapping x' and y' if $x' < y'$, keeping the first operand always greater.

We implement the min-of-max approximation model from Equation 6 in the Pyomo [4] modeling framework. The model is generalized to represent approximations with any number of max-terms, so we model several approximations with various numbers of max-terms, and use the KNITRO solver [5] to optimize each approximation's curve fit. Figure 4 shows a sample optimized approximation with four max-terms. In Section 5.2 we show that seven max-terms is more than enough to achieve reasonable accuracy.

2.2 Negative Numbers and Subtraction

While addition and multiplication give us most of what we need to perform convolution, an additive inverse is required to handle negative constants common to many convolution kernels and complete the mathematical ring. To enable subtraction and negative value representation we adopt an approach similar to memristive crossbars [6] or dual rail computing [44], where we split all numbers into non-negative pairs, $\langle x_{pos}, x_{neg} \rangle$. If the value is positive, x_{pos} equals the value and x_{neg} is 0. For a negative value, x_{pos} is 0 and x_{neg} is the absolute value. For zero, both x_{pos} and x_{neg} are zero.

With this representation, subtraction simply becomes addition with the x_{neg} field of the second operand, but we must re-normalize the result to ensure that at least one of $\langle x_{pos}, x_{neg} \rangle$ is zero. This re-normalization only has to occur at the end of computation and once per convolution. In importance space this re-normalization is achieved with

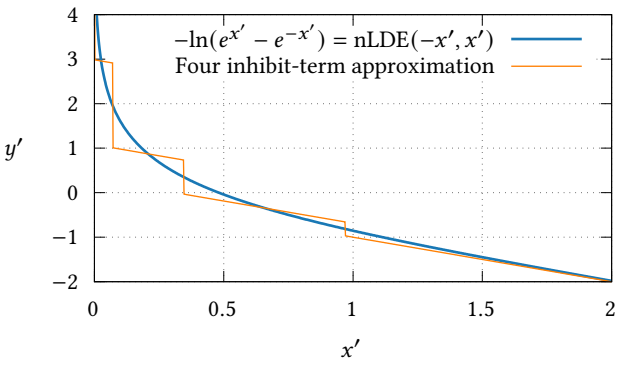


Figure 5. Approximating nLDE with four inhibit-terms. nLDE's shape is more difficult to approximate than nLSE's shape, because nLDE converges to infinity as x' approaches 0, while nLSE converges to $-\ln(2)$ as x' approaches 0 (see Figure 4).

subtraction:

$$x_{pos} = \begin{cases} x_{pos} - x_{neg} & \text{if } x_{pos} \geq x_{neg} \\ 0 & \text{if } x_{pos} < x_{neg} \end{cases}$$

$$x_{neg} = \begin{cases} x_{neg} - x_{pos} & \text{if } x_{neg} \geq x_{pos} \\ 0 & \text{if } x_{neg} < x_{pos} \end{cases}$$

In delay space we re-normalize with the negative log difference exponential function nLDE (Equation 5), which is also symmetric about the y'-axis. The nLDE function is bounded by the inhibit function, similar to how the nLSE function is bounded by min. inhibit(t_i, t_d) accepts two inputs, an inhibiting event set at time t_i and a data event arriving at time t_d, and outputs an event at time t_d if and only if $t_d < t_i$. When $t_d \geq t_i$, no event will be output, which is equivalent to an event at time ∞. We use min-of-inhibit functions to approximate nLDE:

$$\begin{aligned} &\min(\text{inhibit}(x' + E_0, y' + F_0), \\ &\quad \text{inhibit}(x' + E_1, y' + F_1), \\ &\quad \ldots \\ &\quad \text{inhibit}(x' + E_{n-1}, y' + F_{n-1})) \end{aligned} \tag{7}$$

Using the same technique as the nLSE approximation, we simplify from two inputs to one, modeling min-of-inhibit approximations from Equation 7 where $x' + y' = 0$ in Pyomo, and use KNITRO to optimize approximations with various numbers of inhibit-terms. Figure 5 shows a sample optimized nLDE approximation with four inhibit-terms.

2.3 Approximation Logic Design

One of the challenges with a delay-based nLSE function is that its output is always less than or equal to the min of the inputs. When mapped to times, this would mean the output needs to fire before any of the inputs have even arrived and

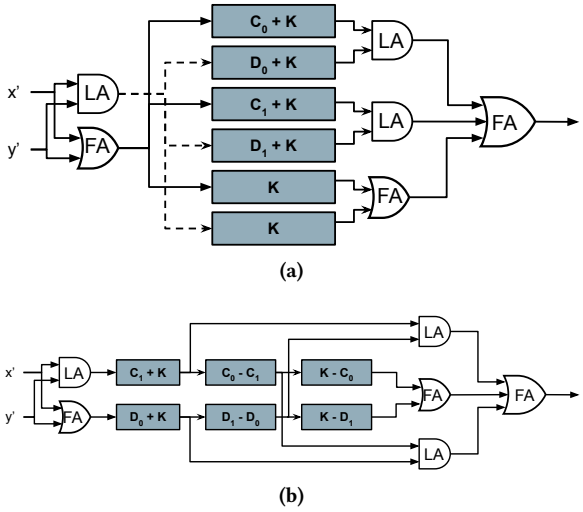


Figure 6. Figure (a) shows A naive circuit implementation of a 2 max-term nLSE approximation where all of the max-terms are calculated in parallel. Figure (b) gives an Optimized hardware implementation of a 2 max-term nLSE approximation using two chains of delay elements. Since the C_i and D_i constants are in opposite order (the largest C_i pairs with the smallest D_i) the output of the first delay element is matched with the last delay element. The grey blocks represent fixed delay elements.

thus the constants in equation 6 are all negative. Instead of directly implementing nLSE, we implement a time shifted nLSE (which is the same as rescaling in importance space). This can be done by adding a constant K that is greater than or equal to the most negative approximation constant:

$$\begin{aligned} &\text{nLSE}(x', y') + K \\ &\approx \min(x', y', \max(x' + C_0, y' + D_0), \ldots) + K \\ &= \min(x' + K, y' + K, \max(x' + C_0 + K, y' + D_0 + K), \ldots) \end{aligned}$$

The now-positive max-term constants can now be implemented with constant delays, and the max and min functions can be replaced with LA and FA gates respectively. Finally, to handle swapping x' and y', as described in Section 2.1, we use a temporal comparator circuit [41] at the input to ensure proper ordering, preventing the need to double number of max-terms. A naive implementation of this can be seen in Figure 6a where each max-term has its own dedicated delay.

However, this approach creates redundant delay, wasting both energy and area. With physical delay elements, energy consumption scales linearly with the magnitude of delay, so designs should minimize the number of redundant delays. This informs an optimized design shown in Figure 6b, where there is only a single path of delay elements for each input. The max-term inputs are tapped from the proper location along the delay chain. Note that as max-terms are added, the

C_i constants increase and the D_i constants decrease, so maxterms are connected in reverse order along their respective chains. This same shift-and-chain approach can be used to create a hardware implementation of the nLDE approximation function (Equation 7).

3 Recurrence Architecture

The accumulation operation in multiply-accumulate implies many connected summations. This can be achieved by all inputs arriving at the same time and being summed together. However, for most systems this is infeasible due to either the number of inputs or data naturally arriving over time. Therefore, a stateful system is required, which poses a problem for delay space. Temporal memories have been proposed in the past but they all rely on either emerging technologies or complex and sensitive analog circuits[26, 29, 39, 45]. To solve this problem we propose an alternative race logic approach that emulates a stateful system for efficient MACs.

This approach relies on the fact that in temporal systems, the arrival time of any event, t_x, is meaningless without context. There must also be a reference time t_{ref}, where the difference between the times $t_x - t_{ref}$ represents x's value. All inputs to race logic operations, including our nLSE approximations, must share the same reference time to function properly. In a system with fully parallelized inputs this is simple with a shared global reference time $t_{ref} = t_{start}$. However, with serialized inputs each input will have a unique local reference time. Before the data can be processed together with race logic it must be synchronized to the last input's reference time. One method of synchronization is delaying each input by the difference between their local reference time and the last input's reference time: $t_{x'} = t_x + (t_{last} - t_{local})$, where an example using nLSE is shown in Figure 7a. These delay lines create state by holding in-flight data until all other inputs have arrived.

This represents an important trick where we can maintain the proper *logical* value of a signal with a shifting reference frame by adding a constant (delay). However, this requires as many delay lines as inputs, which wastes energy and area, making it infeasible for large systems. Instead, we propose an optimization that operates on inputs as they arrive, performing as many operations as possible on currently available data. Then, only the required information for the next stage of computation is passed, minimizing the data that must be reference shifted. This also allows the required reference shifting delay to be reduced because each operation has some inherent delay, either through explicit delay elements or gate propagation time.

Our nLSE approximations are particularly well suited for this compute-on-arrival approach because 1) they have a larger delay due to the addition of a constant K, even further reducing the required frame shift delay, and 2) only accept two inputs. Figure 7b illustrates this by showing how a large nLSE operation can be split up and staged across input arrival times. The equivalence of breaking apart the nLSE operations can be shown simply:

$$\begin{aligned} -\ln(e^{-x} + e^{-y} + e^{-z}) &= -\ln(e^{\ln(e^{-x}+e^{-y})} + e^{-z}) \\ &= -\ln(e^{-\mathrm{nLSE}(x,y)} + e^{-z}) \\ &= \mathrm{nLSE}(\mathrm{nLSE}(x,y), z) \end{aligned}$$

When the calculation being performed is completely associative, commutative and symmetric then each stage is composed of identical hardware, such as our nLSE approximation in Figure 7b. When these conditions are satisfied then the compute-on-arrival approach can be further optimized by frame shifting the output and looping it back into its own input, as shown in Figure 7c. In order for this to operate properly, three constraints must be met:

- The rise and fall time of the delay elements must be matched so that the integrity of the voltage pulse is maintained.
- The value of any input cannot be so large that it extends past the next value's reference frame.
- A relaxation period must be introduced between cycles to ensure that the previous cycle's falling edge does not interfere with the computation of the current cycle.

This *recurrence* takes advantage of reference shifting delay lines and causes the system to act like a classical state machine, so long as the inputs arrive at evenly spaced time intervals. This saves both energy and area by limiting the number and length of delay elements, as well as reducing the number of nLSE approximation hardware blocks required. It's worth noting that this approach is not restricted to a fully serial input. The two-input nLSE approximation units can be expanded to a tree of nLSE blocks with any number of inputs. Regardless of the size of the tree, the output can still be recurrent, creating a trade-off between the number of nLSE approximations, the area of the design, the minimum length of each cycle, and the number of necessary cycles.

4 Rolling Shutter Convolution Architecture

Most cameras rely on a technique called a rolling shutter [17] where individual columns or rows of pixels are captured and read out in parallel. This pipelines both the exposure time and the ADC readout times while causing the inputs to become available across fixed time intervals. This acts as a natural serialization for our recurrence architecture and can be leveraged for convolution. Even in systems without a rolling shutter, a staged readout is often applied to reduce the number of necessary ADCs.

4.1 Analog to Delay Space

Traditional camera systems rely on Analog-to-Digital Conversion (ADC) circuits to read pixel data prior to processing. However, low power systems have explored converting analog signals into the temporal domain [34, 36] for mixed signal

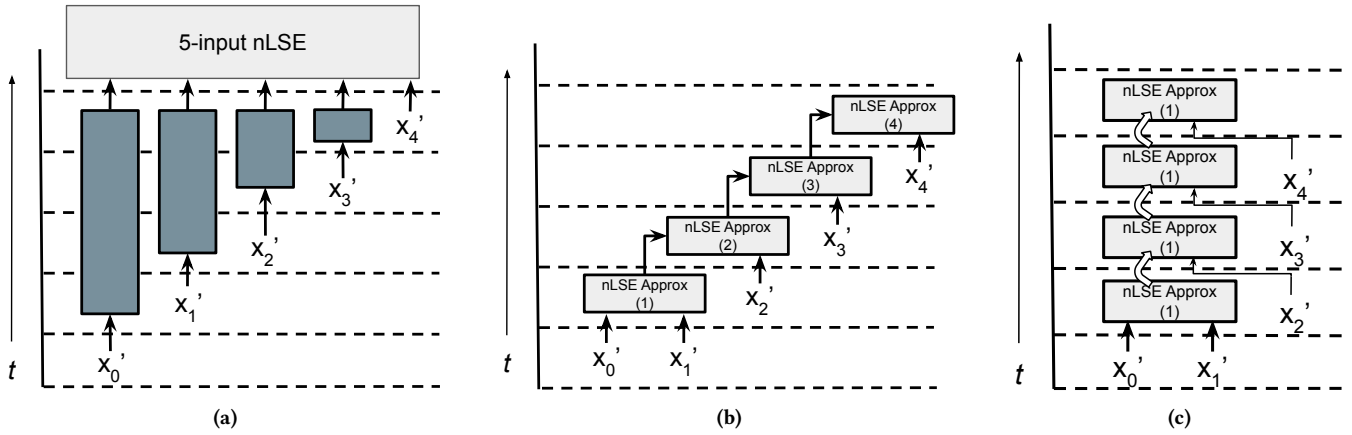


Figure 7. Illustrations of different approaches to reference frame synchronization, where the y-axis indicates time and the x-axis indicates physical area added to the design. Figure (a) shows how each element can be individually delayed to ensure the proper reference frame. Figure (b) shows how computation can be performed as inputs arrive, and Figure (c) shows how the output of this computation can be looped to the same hardware (as shown by the hollow arrow) block with some delay to prevent hardware replication.

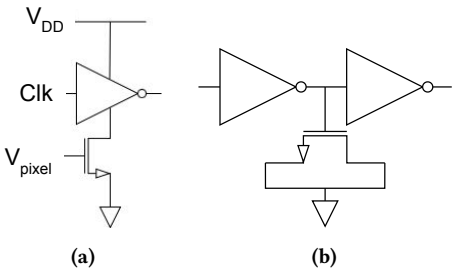


Figure 8. Figure (a) Demonstrates a simple starved inverter that forms the basis of voltage to time converters (VTC). Figure (b) Shows the circuit diagram of an inverter-chain delay element with a transistor between the inverter output and ground to increase the delay of a single inverter.

processing. This approach has shown the potential for significant energy savings [16], and allows for temporal processing of data while in the time domain.

Many of these techniques attempt to achieve a linear mapping of input voltage to output timing [13, 34]. However, this is insufficient for our delay space computation as the negative log of the input is required. Instead, we require a voltage to time converter (VTC) that matches the negative log properties of delay space. This can be achieved by using a starved inverter, shown in Figure 8a, which is already at the core of many VTC systems [16].

The pixel voltage controls the current starving transistor, and the read clock acts as the input to the base inverter. As the pixel voltage increases, the delay of the inverter decreases, creating the value inversion necessary for delay space. Additionally, pixel voltage has a monotonically increasing impact on the delay of the clock signal, allowing it to approximate negative log for specific regions of interest. These two properties combined allow for the delay of the clock signal to be interpreted as a delay space value.

4.2 Delay Elements

Three previous approaches to delay elements for race logic have been proposed: discrete delay through chained DFFs [27], inverter chains, and starved inverters [28]. These approaches all have their advantages, but the discretized approach would be impractical to integrate with the precise constant values required for the delay space approximations.

The starved inverter approach surrounds an inverter with two statically biased transistors [28]. These inverters act as current sources to reduce the rise and fall time of the output, resulting in controllable delays based on the bias voltages. However, this approach requires each starved inverter to be biased separately, introducing circuit complexity and additional energy consumption. Also, having a single large delay element maximizes the impact of random jitter (RJ) from a single element.

Conversely, an inverter chain uses a large number of identical inverters, with the effective load capacitance determining the delay from each individual inverter. In this approach the RJ for each element is independent and scales with the magnitude of delay, causing each additional element to reduce the overall impact of RJ [33]. In our design we use this approach and hard-code the load capacitance by adding a

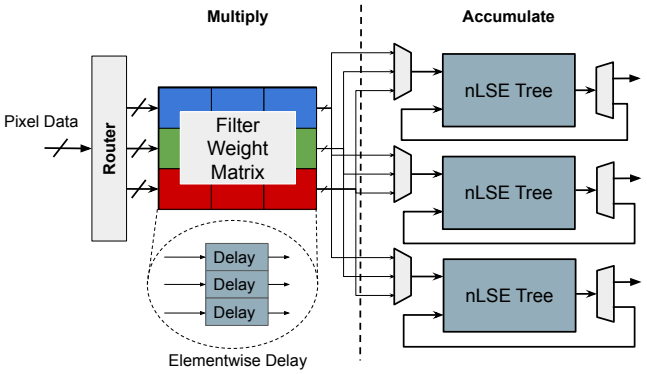


Figure 9. A single, hard-coded, three by three convolution filter. Data is read through the rolling shutter, then progresses through the dedicated filter weight delay matrix and finally to the accumulation nLSE trees. The non-recurrent outputs from each nLSE tree is combined for a single output. The multiply and accumulate hardware handles one row of convolutions and must be replicated for each necessary row.

transistor between the output of each inverter and ground as shown in Figure 8b. The load capacitance is varied by changing the size of the ground transistor, allowing delay to be hard-coded into the hardware. This chained inverter approach introduces an area and noise trade-off, as the larger the output capacitance, the fewer inverters are required for each chain. We evaluate how RJ impacts the accuracy of our approximations in section 5.2, which informs the design of the ground transistor.

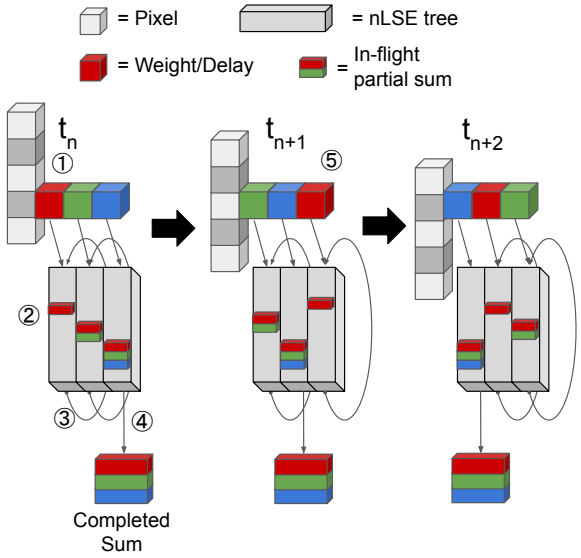
4.3 Dedicated Convolution Engine

The core of this architecture is a hard-coded convolution MAC block shown in Figure 9 which handles all of the convolutions along a set of columns equal to the filter width. This block must be replicated for every filter application along the row axis of the given pixel array, given by $1 + \frac{pixelwidth - filterwidth}{stride}$.

Once the pixel values have been converted to delay space through the VTCs, each temporal pixel value is passed to all MAC blocks that utilize it. Figure 10 illustrates how this pixel data flows through the MAC block. Each block has a static matrix of delay lines that includes every filter weight. As the corresponding pixels are processed by the VTCs they are passed to this matrix and the input can be distributed to any or all of the filter's rows. The number of rows activated in a given cycle is given by $\lceil filterlength/stride \rceil$, where a stride of 1 indicates that every filter row will be used each cycle. ① Each activated filter row then performs element-wise delay, which functions as the multiplication of the MAC operation.

To fully utilize the outputs from the activated rows there must an accumulation unit for every potentially activated filter row. ② These accumulation units receive the results of

Figure 10. An example of the scheduled pixel information flow through the system for a filter with 3 rows. Each differently colored cube represents a different *row* in the filter weight matrix. In subsequent time steps each nLSE tree receives inputs from next filter row until it has completed a full filter application. The actions corresponding to the circled digits are explained in Section 4.3.



the filter rows and subsequently sums the incoming values in delay space. The core of the accumulation unit is a tree of two-input nLSE approximation units performing delay space addition. Whenever the tree is not fully symmetric, gaps in the tree must be path balanced by with a delay equal to the delay of the nLSE approximation hardware to maintain the proper reference frame. This delay is added as deep in the tree as possible to minimize the number of extra delay lines. When an image begins to be processed only one accumulation unit is activated, with the next one being activated every *stride* cycles. Eventually they will all be activated and operating in parallel, similar to a systolic array.

③ The result of this nLSE tree is then looped back as the recurrence discussed in Section 3. This loop must have a delay equal to the cycle time minus the inherent delay of the nLSE tree. ④ Once an nLSE tree has received an input from each of the filter rows it passes its result to the output instead of being looped for recurrence. Then, in the next cycle ⑤ the outputs of the filter rows are rotated so that each nLSE receives the next necessary input for their application of the filter. This ensures that there will be an output produced every *stride* cycles.

4.4 Split Value Representation

The split value representation presented in section 2.2 requires each MAC block to have multiple kernels to handle each combination of {positive, negative} {input, weight}. The

temporal outputs of the VTCs are always non-negative, so we only need two kernels (positive input, positive weight and positive input, negative weight). If the inputs could be negative then we would need a total of four kernels to handle the two additional cases.

Since the weights are hard-coded, the filter weight matrix is split between the positive and negative sides. Weights that are zero in importance space, either because of the pos/neg split or the actual value is zero, become infinity. An infinite delay is the same as the path not existing, allowing the number of weight multiplications (delays) to stay constant with filter size, unaffected by the split value representation. Additionally, the number of nLSE approximations performed remains the same, the only extra operations are any required tree balancing and the recurrence delay lines.

Once a convolution have been completed, both positive and negative kernels must be routed to a delay space subtraction unit to re-normalize the values as described in Section 2.2. The subtraction unit evaluates an nLDE approximation (Section 2.3) on two inputs, and the result can be fed directly to the output. This output can either remain in delay space for further temporal computation, or be converted to the digital domain for traditional processing.

5 Evaluation and Results

5.1 Architectural Simulator

To explore the architectural space we created an architectural simulator that takes a system description as its input and produces a software representation of the architecture. The system description includes the image dimensions, the kernel shape, the number of kernels and the convolution stride. This architecture can be configured to change the unit scale, the maximum supply voltage swing, and the magnitude of each inverter's delay (as a multiplier of the minimum inverter delay). We use the unit scale to indicate the connection between theoretical delay values and physical time values: for example an abstract delay of 1 could map to 5ns.

These parameters are then used to estimate the area and energy consumption for the given system description. The energy estimates are based on SPICE simulations for delay lines using 65nm predictive technology models [49]. Area estimates are dependent on typical transistor sizes for 65nm nodes and an estimation for the total number of transistors in the system. We assume that the delay elements dominate both the energy and area and assume that the control logic is negligible. The architectural parameters are also used to implement noisy versions of delay and our approximations based on noise values from [33].

The generated architecture can be executed given an image data set and filter. The convolution is performed according to the compiled architecture with programmable multiplication, addition and subtraction functions. We use these programmable functions to ensure that when using importance space operations the architecture produces the exact same result as software convolution. We also verify that using exact delay, nLSE and nLDE provides the same result as software convolution when the results are converted from delay space to importance space.

5.2 Approximation Accuracy and Noise

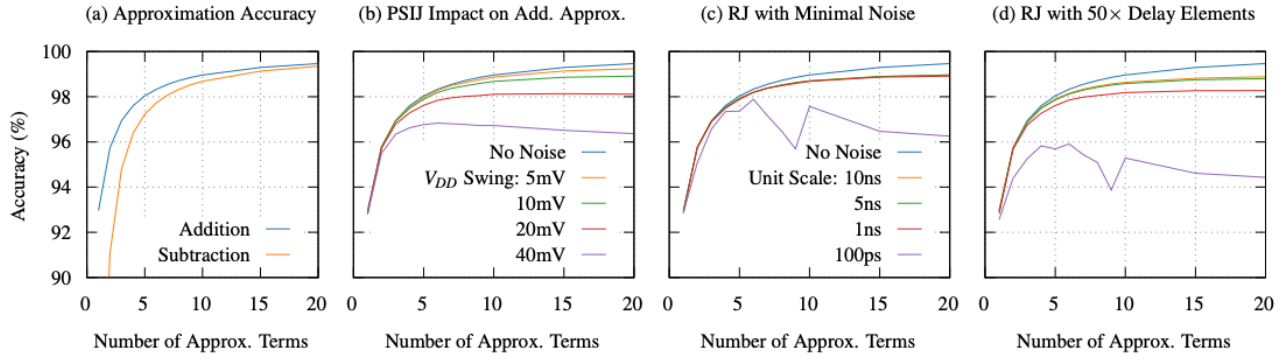
To evaluate our approximation accuracy, we generate two uniform random values between zero and one, which correspond to positive values in delay space. The values are converted to delay space, the approximation is applied, and the result is then converted back to importance space. This approximation result is then compared to the exact operation being performed in importance space. We perform this operation a million times, then take the range-normalized RMS error (RMSE) to determine the overall accuracy.

We use this approach to evaluate how the number of approximation terms impacts the accuracy of our nLSE and nLDE approximations with infinite precision, shown in Figure 12a. The graph shows that additional approximation terms significantly increase the approximation accuracy until 7 or 8 terms where they start to provide diminishing returns. However, when implemented in hardware, the accuracy of the approximation is also impacted by hardware timing noise. We use the noisy approximation simulator described in Section 5.1 to evaluate the two major sources of noise: power supply induced jitter (PSIJ) and random jitter (RJ) [33]. PSIJ is a product of the power delivery network (PDN) and will dominate the noise unless the swing in the supply voltage is carefully controlled. Figure 12b shows how the accuracy of our nLSE approximation suffers due to the noise introduced by varying V_{DD} swings.

While this shows that PSIJ can significantly reduce the accuracy of the approximations, the low power nature of our convolution architecture puts less stress on the PDN, reducing potential voltage swings. If this is still insufficient to control the PSIJ, the voltage swing can be further controlled by adding decoupling capacitors to the PDN. RJ, on the other hand, cannot be controlled and is a function of the magnitude of each inverter's delay [33]. However, since the RJ of each inverter is independent, the more inverters in a chain, the smaller the impact to the system. This creates a relationship between the delay of each inverter, the unit scale, and noise.

Figures 12c and 12d shows the impact of limited PSIJ (10mV V_{DD} swing) and RJ for different unit scales with fixed delay element magnitudes. Figure 12c uses the smallest possible delay for each inverter, minimizing RJ, while Figure 12d uses an inverter with 50× the minimal delay. Both of these show that there is a minimum unit scaling that must be met to ensure the max-terms can be utilized fully, otherwise the RJ noise dominates. With appropriate unit scaling and minimal noise there is little to no impact on the accuracy, but this requires significantly longer inverter chains to achieve the proper delay. However, Figure 12d shows that for a unit

Figure 11. Delay space approximation accuracy as the number of approximation terms increases. Figure (a) shows the pure approximation accuracy, while Figure (b) shows how PSIJ impacts accuracy. Figure (c) and figure (d) show how RJ impacts accuracy. All graphs share the same y-axis, and (c) and (d) share the same legend. The top line is the same across all graphs, showing delay space approximation accuracy for addition, without noise.



scale of 5ns the size of the inverter chains can be cut by 50× with minimal noise impact. Beyond 5ns the hardware approximation improves slightly, but may not justify the extra energy required for the longer unit scaling.

Surprisingly the impact of noise, regardless of source, is larger as more approximation terms are added. Each additional term reduces the difference between approximation constants (C_i and D_i in Section 2.1), which increases the probability that the FA gate selects the wrong approximation term due to noise. The incorrect approximation term will then have a larger impact on the output than slight variations along the proper max-term.

Note that the nLDE approximation is also affected by noise, but because there is a larger difference between its approximation constants, the noise impacts the accuracy to a lesser degree. Due to space constraints we omit the nLDE noise trade-off graphs from this paper, but we consider this impact on nLDE accuracy in our architectural evaluation.

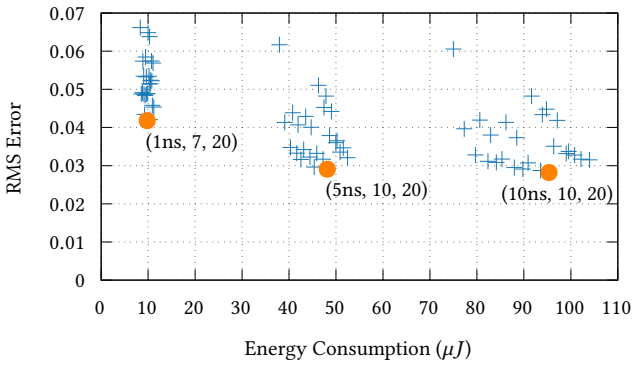


Figure 12. The results of the architectural exploration where the number of approximation terms are swept across 5, 7, 10, 15, 20 for both nLSE and nLDE and the unit scale is swept across 1ns, 5ns and 10ns. Circled points are along the Pareto optimal frontier, with the corresponding configuration (Unit scale, nLSE terms, nLDE terms) written next to it.

5.3 Architectural Evaluation

To investigate the relationship between approximation accuracy and the our convolution architecture, we run a design space exploration with our architectural simulator. For our exploration we use the Imagenette [18] dataset, which is a subset of the Imagenet [11] dataset, and scale each image to 150 by 150 pixels. Our architecture is then configured to run the Sobol function from OpenCV [2] which uses two 3×3 filters. We sweep the number of approximation terms for both nLSE and nLDE as well as the unit scale. The delay of each inverter is set to 50× the minimal delay and the maximum V_{DD} swing is set to 10mV. For each configuration, the architectural simulator determines the energy consumption. Then it emulates all of the operations with appropriate noise in the same order as the simulated hardware for a single channel of the dataset. The Sobel convolution was calculated for five different images and the RMSE for each output pixel was taken. The results from this architectural explorations are shown in Figure 12.

The first noteworthy result from this graph is the effect of unit scale. Each vertical grouping represents a unique unit scale, caused by the necessary increase in delay magnitudes, which leads to a larger energy consumption. Also, the energy difference between different approximation configurations becomes more significant as the unit scale increases. This is because the increased delay due to additional approximation terms is multiplied by the larger unit scale, leading to more energy consumption.

As expected, the accuracy increases noticeably when going from 1ns to 5ns unit scales. However, similar to the two input accuracy in Figure 11, the difference between 5ns and

Table 1. Convolution benchmarks and their descriptions. The filter configuration is (filter size, stride, number of filters).

Function	Description	Filter Config
Sobel	Edge Detection	3×3, 1, 2
pyrDown	Blur and Downsample	5×5, 2, 1
GaussianBlur	Blur with Gaussian filter	7×7, 1, 1

Table 2. Relevant statistics for different race logic convolution configurations for three different functions. The Arch column indicates the unit scale, the number of max-terms and the number of inhibit-terms. The pyrDown and GaussianBlur functions do not have inhibit-terms as they have only positive filter weights. Throughput is measure in millions of frames per second (Mfps)

Function	Arch.	Area (mm^2)	Energy per Frame (μJ)	Max Through-put ($Mfps$)	Acc. ($RMSE$)
Sobel	1ns, 7, 20	.02	9.81	71	.065
	5ns, 10, 20	.08	48.1	18	.029
	10ns, 10, 20	.149	95.4	9	.028
pyrDown	1ns, 7	.004	7.2	55	.038
	5ns, 10	.134	36.6	12	.029
	10ns, 10	.236	72.7	6	.028
GaussianBlur	1ns, 7	.008	14.2	55	.037
	5ns, 10	.273	73.1	12	.028
	10ns, 10	.481	146	6	.027

10ns is fairly small. This demonstrates an effect where increasing the unit scale improves the accuracy with diminishing gains while the energy continues to increase consistently. It is clear from the Pareto frontier that maximizing the number of inhibit terms is important for accuracy. As discussed earlier in Section 5.2, inhibit terms are less affected by noise. Also, adding additional inhibit terms leads to a smaller increase in the energy consumption compared to adding max-terms because there are significantly fewer nLDE operations. Due to the larger impact of noise on the nLSE approximation, increasing the number of max-terms beyond seven or ten does not significantly improve the accuracy.

In Figure 12 we highlight three configurations that are along the Pareto optimal frontier and use these configurations to investigate the functions shown in Table 1 from the openCV library [2]. Note that the pyrDown and GaussianBlur functions have non-negative weight values, so the split value representation and nLDE approximation unnecessary. Table 2 shows the area, energy and cycle time of these functions for each of the highlighted configurations. The Sobel function is the only one with negative weights, requiring the split value representation. However, it still consumes a similar amount of resources to pyrDown due to the fact that it has significantly smaller filter sizes. While pyrDown consumes marginally less energy than the Sobel function, this is largely due to the fact that its stride size is two, effectively halving the number of necessary computations and static hardware. GaussianBlur demonstrates that larger filter sizes consume significantly more resources, doubling the energy and area requirements from pyrDown while barely increasing the accuracy. However, because the height of the nLSE tree is the same for both functions, the maximum possible throughput is the same.

The maximum possible throughput for all of these architectures is incredibly high, with the fastest filter able to process 71 million frames-per-second (FPS), although it is unlikely that a camera would able to match this throughput. So in a slower camera the delay space architecture will be able process the convolutions without slowing down the throughput of the system. The only requirement is that the row readout cycle time be evenly spaced and short enough for recursion be energy efficient. If this means the readout is faster than the rest of the camera operations then the time between finishing an image and starting to read the next can be lengthened to match the rest of the system's FPS.

This shows how delay space convolutions can be used for a variety of workloads with low energy and no impact to the processing speed. To see how this impact compares to a state of the art approach we consider a processing-in-pixel (PIP) architecture [23] as shown in Table 3. For our comparison we evaluate the benchmark used in the PIP paper, a 1.5 bit edge detection convolution where the weights can be zero, 1 or -1. We evaluated our design against three different filter sizes with two different strides. For the calculation of our energy we include both the VTC [13] and TDC [14] cost, and use a 1ns unit scale, 10 max-terms and 20 inhibit-terms delay space configuration. We report the energy per pixel per frame as the same figure of merit presented by the PIP work. We also ran the same convolution benchmark as the PIP in our simulator, and normalized our RMSE to the range of output values to create a percent error for direct comparison.

The delay space approach consumes less energy than PIP for every convolution structure when the results are left in the temporal domain. As the convolution gets larger and the stride stays small, the energy improvements of the delay space architecture becomes more significant, achieving over 2.5× energy savings for a four by four filter with a stride of two. The delay space architecture also outperforms PIP when the stride size is small, but PIP improves its energy consumption more with an increasing stride size than delay space. This makes delay space a better fit for most openCV filtering applications, where the stride size normally always small. However, when the results must be converted into digital the delay space architecture begins to consume more energy for the ultra-low energy design points. This motivates additional computation in the temporal domain, such as more convolutional layers or min/max selections.

Table 3. Comparison between the delay space convolutions and a state of the art, processing in pixel convolution engine. We show the energy of the operation, the delay of each operation, the accuracy and the energy delay product. The delay space convolution analysis includes the energy cost of the VTC, and separate columns are given for a system that includes the TDC cost. The minimum energy delay product for each configuration is shown in bold.

Benchmark		PIP				Delay Space					
Shape	Stride	Energy per pixel per frame (pJ)	Frame Delay (mS)	E×D Product ($pJ \times mS$)	Error ($\%RMSE$)	Energy per pixel per frame (pJ)	Energy w/TDC (pJ)	Min Frame Delay (mS)	E×D Product ($pJ \times mS$)	E×D Product w/TDC ($pJ \times mS$)	Error ($\%RMSE$)
2x2	2	16.9	12.8	2.18e2	7.18	16.4	21.9	7.35e-4	**1.21e-2**	1.61e-2	3.69
2x2	4	4.6	5.2	2.39e1	7.12	4.2	9.8	7.35e-4	**3.13e-3**	7.2e-3	3.51
2x4	2	32.9	21.9	7.21e2	7.8	21.3	26.8	7.35e-4	**1.57e-2**	1.97e-2	3.02
2x4	4	7.0	7.7	5.42e1	6.77	5.46	11.0	7.35e-4	**4.01e-3**	8.09e-3	3.6
4x4	2	104	41.3	4.29e3	4.56	41.0	46.6	1.47e-3	**6.04e-2**	6.86e-2	2.8
4x4	4	11.6	1.3	1.52e2	5.27	10.3	15.9	1.47e-3	**1.52e-2**	2.34e-2	3.2

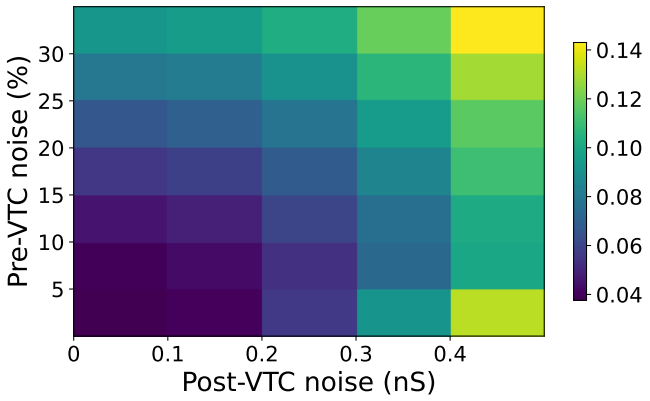


Figure 13. Impact of sensor and VTC noise on the accuracy of the pyrDown convolution. The axes show the standard deviation of the introduced error while the color indicates the output RMSE. The y-axis is given as a percentage of the max possible range of input values while the x-axis is given in nanoseconds.

Also, the delay space architecture approach can be completely separated from the pixel array, unlike PIP. PIP requires circuitry to be added to the pixel array, significantly reducing the possible pixel density and the speed of image processing. This causes the energy delay product of the PIP architecture to be several orders of magnitude higher than the delay space architecture, which operates fast enough that it has minimal impact on the image throughput.

5.4 VTC noise impact

We've shown that our proposed delay space arithmetic is able to process convolutions with incredibly low energy while still maintaining good accuracy, but this evaluation assumes a perfect sensor and VTC. In reality, these systems are noisy, which can contribute to error in the final result of the delay space convolutions. This noise falls into two general categories, noise that occurs before the VTC and noise at the VTC output. Noise that occurs before the VTC are associated with the CMOS sensor, such as fixed pattern noise [15] and dark shot noise [12], and affects the voltage used to program the starved inverter. The resulting value will then undergo a negative log transformation with the conversion to delay space. Noise that occurs after the VTC is caused by nonidealities in the VTC, and results in a linear change in the delay space value.

To evaluate the impact of these two types of noise we use the pyrDown application from OpenCV [2]. We use the same testing framework and dataset as before, but instead we add a random Gaussian distribution to each image centered around zero at two different points, before the negative log conversion to delay space, and after. For both types of noise we gradually increase the standard deviation of the distribution. The noise is applied in simulation, and represents an abstract noise model to show how robust the system is to noise. For the delay space arithmetic we use a 1ns unit scale, a 10 max-term system with a maximum V_{DD} swing of 10mV and the same noise model as before. The results of this experiment are shown in Figure 13. The y-axis indicates noise on the original, importance space image, where the values are the standard deviation of the noise, shown as a percentage of the maximum input range. The x-axis shows the noise applied after the conversion to delay space, where each value is the standard deviation of the noise in nanoseconds. The color or each "pixel" corresponds to the output RMSE from an ideal convolution with no error and noise.

As expected, the RMSE increases with the noise being added to the system. However, the error of the delay space computation grows slower than the amount of noise. With a noise distribution that has a standard deviation of 10% of the input range and no post-VTC noise, the RMSE is .046, less than .01 greater than the baseline which assumes a perfect input. Also, the post-VTC noise has a small impact until it

reaches a standard deviation of .3 ns. This is because the post-VTC noise is added in the log domain, which means it has an exponential impact on the importance space input. So small errors have very little impact, but as the post-VTC noise increases the error increases exponentially.

This exponential growth is most apparent in the case when there is no pre-VTC noise. The RMSE grows faster along this row than any others, and has almost as much error as the maximum noise data point. This is because both noise sources are added independently, so the two sources of noise can partially counteract each other. However, in a real system there will always be some noise both before and after the VTC. Overall this chart shows that the delay space architecture is fairly robust to noisy inputs and can tolerate some non-idealities caused by the VTC.

6 Related Work

Historically, work reducing the overheads of bit-parallel computation has centered around bit-serial computation. Recently this has been used to create energy efficient MACs [8], solving partial differential equations [35], and neural networks [21]. Memory cells have even been adapted to perform bit-serial computation near data [21, 37]. However, bit-serial computation replaces the bit-parallel computation with many cycles and high activity factors. It also requires registers and memory to keep track of data across cycles, increasing the area and energy consumption.

Similar to bit-serial computing, stochastic computing encodes information into a probabilistic bit-stream. Now computation can be done using just AND and OR gates [1]. Because random number generators are expensive researchers have investigated methods to reduce the dependence on independent streams [47]. This work has then shown how it can be applied in low power, near-sensor domains such as the brain [46]. However, this still requires a large number of cycles and expensive averaging circuitry.

Race logic has been proposed previously as a low power alternative to bit-serial and stochastic computing, and has shown to be very effective for dynamic programming [28] and decision trees [43]. However, our work is the first to propose an arithmetic framework for race logic, to the best of our knowledge. There exists another form of time-domain computation that uses pulse width to encode information [32]. General addition has been shown to be possible with time registers [7, 40], and multiplication has been shown by scaling this technique [39]. However, these operations work through iterative shifting and adding, requiring each input to fully complete before the next can begin. Also, these time registers have temporal limits and overflow issues.

Along with these strategies many analog hardware approaches have been coupled with image sensors. Convolution circuits have been placed alongside the photo diode for processing-in-pixel [22]. However, this approach has less accuracy than traditional approaches and causes the pixel size to increase significantly, reducing the resolution of cameras using this technique. Mixed near-sensor and in-sensor architectures have been proposed [25], but the near-sensor computation is based on conventional binary computation.

Race logic has been applied to 3D photon cameras [19] to reduce off-sensor bandwidth and computation. However, this is done using race logic to cleverly find the median without doing any actual arithmetic. Similarly, a time domain approach has been used in a retinal prosthesis [9] to perform energy efficient edge detection. They compare pulse widths with neighboring pixels and use mixed signal approaches to create a threshold for the differences. While this is similar to edge detection convolution (the Sobel operation), it cannot be generalized to other filters. We expect the new arithmetic capabilities of this energy efficient approach might open the door to even more applications in the future.

7 Conclusion

There is no question that convolution operations will continue to play an important role in sensor information processing, with applications from image processing to object detection and neural networks. The power consumption of multiply-accumulate (MAC) operations is a key factor in convolutions integrated with sensor operation. Departing from the norm of performing full analog-to-digital conversions for each pixel, we showed how to perform arithmetic on temporally encoded signals with remarkable energy efficiency.

At the heart of our approach is a negative log transformation, converting the traditional numeric space into a 'delay space'. This mechanism enables multiplicative scaling by adding delays. We demonstrate the direct execution of negative log-space addition and subtraction in this new delay space, ensuring that normal associative and communicative properties of addition still apply in the transformed operations. Moreover, we show how strong approximations of these operations can be efficiently constructed from delay elements and existing CMOS logic elements.

Many computations execute iteratively, and to apply temporal techniques we need new techniques for chaining and recurrently operating these designs, multiplexing in both time and space. We show how time-division multiplexing aligns naturally with the staged ADC readout common to most modern sensor arrays. To establish the practicality of this approach, we present an automated transformation that carries traditional convolutions through to their delay space equivalents. This translation balances error introduced by a new temporal equivalent of quantization and delay element noise. We use this approach to show how our approach can consume eight times less energy than another state-of-the-art convolution approach while achieving similar accuracy. We believe that this approach presents a powerful new set of design primitives with applications beyond convolution.

Acknowledgments

The authors would like to thank Michael Beyeler, Sara Achour, and all of the anonymous referees for their valuable comments and helpful suggestions. This work is supported by the National Science Foundation under Grant No. 1763699, 1730309, and 1717779.

References

[1] Armin Alaghi and John P Hayes. Survey of stochastic computing. *ACM Transactions on Embedded computing systems (TECS)*, 12(2s):1–19, 2013.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] John S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

[4] Michael L. Bynum, Gabriel A. Hackebeil, William E. Hart, Carl D. Laird, Bethany L. Nicholson, John D. Siirola, Jean-Paul Watson, and David L. Woodruff. *Pyomo–Optimization modeling in Python*, volume 67. Springer Science & Business Media, third edition, 2021.

[5] R. H. Byrd, J. Nocedal, and R.A. Waltz. *KNITRO: An integrated package for nonlinear optimization.* Large-Scale Nonlinear Optimization. Springer, 2006.

[6] Weidong Cao, Xin He, Ayan Chakrabarti, and Xuan Zhang. NeuADC: Neural network-inspired RRAM-based synthesizable analog-to-digital conversion with reconfigurable quantization support. In *Design, Automation and Test in Europe Conference (DATE)*, pages 1477–1482, 2019.

[7] Zhengyu Chen and Jie Gu. 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):624–635, 2021.

[8] Harsh Chhajed, Gopal Raut, Narendra Dhakad, Sudheer Vishwakarma, and Santosh Kumar Vishvakarma. Bitmac: Bit-serial computation-based efficient multiply-accumulate unit for DNN accelerator. *Circuits, Systems, and Signal Processing*, pages 1–16, 2022.

[9] Dong-Hwi Choi and Dong-Woo Jee. A 1984-pixels, 1.26 nW/pixel retinal prosthesis chip with time-domain in-pixel image processing and bipolar stimulating electrode sharing. *IEEE Journal of Solid-State Circuits*, pages 1–10, 2023.

[10] Daniel G. Costa. Visual sensors hardware platforms: A review. *IEEE Sensors Journal*, 20(8):4025–4033, 2020.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[12] Oliver E Dial. Ccd performance model. In *Surveillance Technologies*, volume 1479, pages 2–11. SPIE, 1991.

[13] Ahmed Elgreatly, Ahmed Dessouki, Hassan Mostafa, Rania Abdalla, and El-sayed El-Rabaie. A novel highly linear voltage-to-time converter (VTC) circuit for time-based analog-to-digital converters (ADC) using body biasing. *Electronics*, 9(12):2033, 2020.

[14] Ryuichi Enomoto, Tetsuya Iizuka, Takehisa Koga, Toru Nakura, and Kunihiro Asada. A 16-bit 2.0-ps resolution two-step TDC in 0.18-μ m CMOS utilizing pulse-shrinking fine stage with built-in coarse gain calibration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(1):11–19, 2018.

[15] Peter W Fry, Peter JW Noble, and Robert J Rycroft. Fixed-pattern noise in photomatrices. *IEEE Journal of Solid-State Circuits*, 5(5):250–254, 1970.

[16] Ali H Hassan, Hassan Mostafa, Tawfik Ismail, and SRI Gabran. An ultra-low power voltage-to-time converter (VTC) circuit for low power and low speed applications. In *29th IEEE international system-on-chip conference (SOCC)*, pages 178–182. IEEE, 2016.

[17] Gerald C. Holst and Terrence S. Lomheim. *CMOS/CCD sensors and camera systems.* SPIE Press Monograph, 2007.

[18] Jeremy Howard. Imagenette. https://github.com/fastai/imagenette/.

[19] Atul Ingle and David Maier. Count-free single-photon 3d imaging with race logic. *arXiv preprint arXiv:2307.04924*, 2023.

[20] Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A study of BFLOAT16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

[21] Hyunjoon Kim, Taegeun Yoo, Tony Tae-Hyoung Kim, and Bongjin Kim. Colonnade: A reconfigurable SRAM-based digital bit-serial compute-in-memory macro for processing neural networks. *IEEE Journal of Solid-State Circuits*, 56(7):2221–2233, 2021.

[22] Martin Lefebvre, Ludovic Moreau, Rémi Dekimpe, and David Bol. 7.7 a 0.2-to-3.6 TOPS/W programmable convolutional imager SoC with in-sensor current-domain ternary-weighted MAC operations for feature extraction and region-of-interest detection. In *IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 118–120. IEEE, 2021.

[23] Martin Lefebvre, Ludovic Moreau, Rémi Dekimpe, and David Bol. 7.7 a 0.2-to-3.6TOPS/W programmable convolutional imager SoC with in-sensor current-domain ternary-weighted MAC operations for feature extraction and region-of-interest detection. In *IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 118–120, 2021.

[24] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 2021.

[25] Tianrui Ma, Yu Feng, Xuan Zhang, and Yuhao Zhu. CAMJ: Enabling system-level energy modeling and architectural exploration for in-sensor visual computing. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*, ISCA '23, New York, NY, USA, 2023. Association for Computing Machinery.

[26] Advait Madhavan, Matthew W. Daniels, and Mark D. Stiles. Temporal state machines: Using temporal memory to stitch time-based graph computations. *J. Emerg. Technol. Comput. Syst.*, 17(3), may 2021.

[27] Advait Madhavan, Timothy Sherwood, and Dmitri Strukov. Race logic: A hardware acceleration for dynamic programming algorithms. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA)*, ISCA '14, page 517–528. IEEE Press, 2014.

[28] Advait Madhavan, Timothy Sherwood, and Dmitri Strukov. A 4-mm2 180-nm-CMOS 15-giga-cell-updates-per-second DNA sequence alignment engine based on asynchronous race conditions. In *IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4, 2017.

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

[30] N.R. Mahapatra, A. Tareen, and S.V. Garimella. Comparison and analysis of delay elements. In *The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002.*, volume 2, pages II–II, 2002.

[31] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *21st International Conference on Artificial Neural Networks (ICANN)*, pages 52–59. Springer, 2011.

[32] Daisuke Miyashita, Shouhei Kousai, Tomoya Suzuki, and Jun Deguchi. Time-domain neural network: A 48.5 TSOp/s/W neuromorphic chip optimized for deep learning and CMOS technology. In *IEEE Asian Solid-State Circuits Conference (A-SSCC)*, pages 25–28. IEEE, 2016.

[33] Xunjun Mo, Jiaqi Wu, Nijwm Wary, and Tony Chan Carusone. Design methodologies for low-jitter CMOS clock distribution. *IEEE Open Journal of the Solid-State Circuits Society*, 1:94–103, 2021.

[34] Hassan Mostafa and Yehea I Ismail. Highly-linear voltage-to-time converter (VTC) circuit for time-based analog-to-digital converters (T-ADCs). In *IEEE 20th international conference on electronics, circuits, and systems (ICECS)*, pages 149–152. IEEE, 2013.

[35] Junjie Mu and Bongjin Kim. 29.2 a 21× 21 dynamic-precision bit-serial computing graph accelerator for solving partial differential equations using finite difference method. In *IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 406–408. IEEE, 2021.

[36] Holly Pekau, Abdel Yousif, and James W Haslett. A cmos integrated linear voltage-to-pulse-delay-time converter for time based analog-to-digital converters. In *IEEE International Symposium on Circuits and Systems*, pages 4–pp. IEEE, 2006.

[37] Xiangjun Peng, Yaohua Wang, and Ming-Chang Yang. Chopper: A compiler infrastructure for programmable bit-serial SIMD processing using memory in DRAM. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1275–1288. IEEE, 2023.

[38] Stephen J Sangwine and Todd A Ell. Colour image filters based on hypercomplex convolution. *IEE Proceedings-Vision, Image and Signal Processing*, 147(2):89–93, 2000.

[39] Aseem Sayal, Shirin Fathima, SS Teja Nibhanupudi, and Jaydeep P. Kulkarni. 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):2205–2220, 2021.

[40] Aseem Sayal, S. S. Teja Nibhanupudi, Shirin Fathima, and Jaydeep P. Kulkarni. A 12.08-TOPS/W all-digital time-domain CNN engine using bi-directional memory delay lines for energy efficient edge computing. *IEEE Journal of Solid-State Circuits*, 55(1):60–75, 2020.

[41] James Smith. Space-time algebra: A model for neocortical computation. In *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 289–300. IEEE, 2018.

[42] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

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

[44] Georgios Tzimpragos, Jennifer Volk, Alex Wynn, James E Smith, and Timothy Sherwood. Superconducting computing with alternating logic elements. In *International Symposium on Computer Architecture (ISCA)*, pages 651–664. IEEE, 2021.

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

[46] Di Wu, Jingjie Li, Zhewen Pan, Younghyun Kim, and Joshua San Miguel. uBrain: A unary brain computer interface. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*, pages 468–481, 2022.

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

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

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

[50] Djemel Ziou, Salvatore Tabbone, et al. Edge detection techniques-an overview. *Pattern Recognition and Image Analysis C/C of Raspoznavaniye Obrazov I Analiz Izobrazhenii*, 8:537–559, 1998.