

SCIMITAR: Stochastic Computing In-Memory In-Situ Tracking ARchitecture for Event-Based Cameras

Wojciech Romaszkan¹, Jiyue Yang¹, Alexander Graening¹, *Graduate Student Member, IEEE*,
Vinod K. Jacob¹, Jishnu Sen¹, Sudhakar Pamarti¹, *Senior Member, IEEE*, and Puneet Gupta¹, *Fellow, IEEE*

Abstract—Event-based cameras offer low latency and high-dynamic range imaging data in a sparse format that is well-suited for high-speed object tracking. Processing this sparse data in the same way as traditional camera data requires a great deal of unnecessary computation, making it difficult to take advantage of the high-effective frame rate for real-time processing. In this work, we propose an accelerator for high-speed object tracking on event-based camera data. SCIMITAR combines digital in-memory stochastic computing, in-situ stochastic stream generation, and multiple optimizations for utilizing input sparsity. SCIMITAR provides unparalleled performance with latency and energy that scale with sparsity. We demonstrate SCIMITAR performance on an object tracking application using circuit-level simulations of custom-designed compute-in-memory (CIM) macros and digital circuits. We achieve a frame processing rate of 26k frames/s with 100 regions-of-interest per frame and equivalent or better than state-of-the-art tracking accuracy. The accelerator achieves a peak throughput of 71 TOP/S and energy efficiency of 733 to 1702 TOP/S/W demonstrated on a range of event-based vision datasets, which is 5× higher than other CIM solutions.

Index Terms—Approximate computing, computer vision, gabor filters, in-memory computing, neuromorphic cameras, object tracking, stochastic computing, systems architecture.

I. INTRODUCTION

EVENT-BASED cameras [1] transmit information about brightness changes as an asynchronous event stream. The characteristics of these cameras make them preferable to frame-based cameras for applications, such as object tracking [1], [2].

Event-based cameras generate low latency, high-data-rate outputs that enable tracking high-velocity objects without the motion blur that plagues conventional cameras [3]. However, general-purpose processing architectures cannot deal with this low latency and utilize the sparsity presented by this data, making a strong case for custom accelerators for event-based data [4], [5].

Manuscript received 12 August 2024; accepted 13 August 2024. Date of current version 6 November 2024. This work is based on research funded by Northrup Grumman and the DARPA FRANC/ERI-DA program. This article was presented at the International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS) 2024 and appeared as part of the ESWEEK-TCAD Special Issue. This article was recommended by Associate Editor S. Dailey. (*Corresponding author: Alexander Graening.*)

The authors are with the Department of Electrical and Computer Engineering, University of California, Los Angeles, CA 90095 USA (e-mail: wromaszkan@ucla.edu; jyang669@ucla.edu; agraining@ucla.edu; jvinod@ucla.edu; jishnu@ucla.edu; spamarti@ee.ucla.edu; puneetg@ucla.edu).

Digital Object Identifier 10.1109/TCAD.2024.3448227

To push the performance of event-based object tracking, we propose SCIMITAR: stochastic computing (SC) in-memory in-situ tracking architecture for event-based cameras. SC [6], [7], [8] uses logic gates as basic compute units along with several optimizations to achieve massive parallelism and extremely high efficiency for sparse data. Compute-in-memory (CIM) [9], [10], [11] embeds computation inside the memory to reduce data movement and significantly improve energy efficiency. SCIMITAR combines the benefits of SC and CIM by using compact and efficient in-situ stochastic number generators (SNGs) in the memory to overcome the difficulty of converting binary to stochastic numbers. Although prior work has proposed stochastic CIM (SCIM) by storing preconverted stochastic numbers, it requires spatial unrolling of SC streams, which can lead to a large chip area [10]. We show, through the use of analytical modeling and detailed simulations, that SCIM with in-situ SNG can deliver unprecedented energy efficiency for processing event-based data through combined innovations in microarchitecture and circuit design. Our contributions are as follows: 1) we demonstrate that a tracking pipeline with event-based cameras based on low-bitwidth Gabor-filter can achieve state-of-the-art accuracy; 2) the first accelerator architecture for event-based object detection and tracking using scalable in-situ SCIM; 3) scalable in-situ SCIM processing with performance 5× higher than state-of-the-art; 4) a set of microarchitecture techniques, including channel load skipping, zero detection, and column maximum tracking, co-designed with the SCIM, to support input sparsity (1.54× energy-efficiency improvement), coupled with dynamic early termination (ET) scheme (62.8% latency and energy reduction); and 5) 733 to 1702 TOP/S/W for sparse inputs demonstrated on a range of event-based datasets.

II. MOTIVATION

A. Event-Based Cameras

Event-based cameras are often referred to as neuromorphic since their design and data format were inspired by the human eye [1]. The biological retina produces spikes as individual cells sense changes in intensity. Similarly, event-based cameras only transmit pulses, or events, when the brightness of a given pixel changes. The output from such a camera is a stream of positive and negative events depending on the polarity of the change. For a stationary camera, moving objects will cause events, but background objects will not, thus highlighting the most critical information in the scene

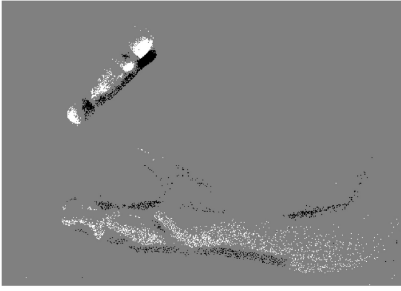


Fig. 1. Spinning marker. This image shows data accumulated over a 1-ms interval, generated from tossing a marker in the air in front of a cluttered background, which is filtered out by the camera. The white events indicate an increase, and the black ones indicate a decrease in brightness.

for applications, such as tracking [1], [12], [13]. Fig. 1 shows how moving objects appear while the background disappears. If the event-based camera is moving, it will highlight the edges of objects, which can be useful for applications, such as localization [1], [14], [15]. Advantages of event-based cameras include the following.

- 1) *Sparse Data Output*: Only changing pixels transmit events, significantly reducing bandwidth compared to transmitting the entire frame [1].
- 2) *Fast Response to Changes*: Information about changing pixels is transmitted immediately since there is no concept of frame rate. Pixel latency varies from 3 to 120 μ s. [1], [16].
- 3) *High-Dynamic Range*: Pixels operate independently and respond to changes in the log of intensity, so event-based cameras can have dynamic range on the order of 130-140 dB compared to 60 dB for conventional cameras [1], [16], [17].
- 4) *Very High-Temporal Resolution*: Existing event-based cameras support between 1 and 1200 mega-events per second (MEPS) [1], [16], providing useful information comparable to $> 10,000$ frames/s (fps) on a frame-based camera which is typically 30-120 fps.

B. Event-Based Data Processing

1) *Background*: Data from an event camera typically consists of a string of x -location, y -location, polarity, and sometimes timestamp for each event referred to as address-event-representation (AER). Given this data format, we must determine the best way of processing such information to guide our architecture choice. Most approaches fit into two groups [1]. First are the algorithms that process individual events [4], [18]. Those *event-based* methods update their state on every incoming event, guaranteeing minimum latency while avoiding the processing of irrelevant data. The second type of approach processes events in groups [3], [19]. These groups are typically sparse, reconstructed frames, so we refer to those algorithms as *frame-based*. The frame format makes it possible to employ well-established computer vision (CV) algorithms that are difficult to apply to data event-by-event in AER.

The required compute decreases if we process regions-of-interest (ROIs) [20], [21], [22] or parts of the field of view where we expect objects to be instead of processing the entire frame. For frame-based processing, using ROIs allows us to ignore large parts of the frame while still being able to harness

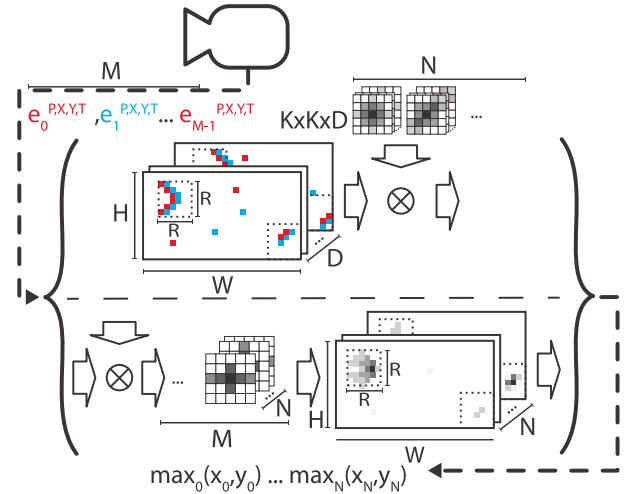


Fig. 2. Analytical model parameters for frame-based (top) and event-based (bottom) processing.

TABLE I
ANALYTICAL MODEL METRICS

	Input	Accesses Output	Weight	MAC
Event	$2M$	K^2MN	K^2DN	K^2MN
Frame	$M+R^2DC$	CN	K^2DN	K^2DR^2CN

conventional CV methods. ROI detection, or *region proposal*, requires less energy and is faster than convolution [23], [24]. Hence, we assume its use in all further considerations for our architecture.

At first glance, an event-based method seems like it should be more efficient than a frame-based method due to minimizing computations, but this does not account for memory access requirements, so we conducted the following analysis that shows when frame reconstruction is better.

We start with a few assumptions. In one iteration, we process C ROIs of size $R \times R$ containing M events. W and H are the frame width and height and generally $R \ll W, H$. We divide the duration t of data collection into D bins of events. We use N spatiotemporal filters of the size $K \times K \times D$, convolved with the image data. The maximum output of each filter is recorded and used for tracking. Section II-C describes our assumed algorithm. We assume the computational cost of selecting ROIs is negligible as a tracking byproduct. All parameters of our model are shown in Fig. 2.

We consider the performance metrics in Table I. For simplicity, we assume all memory accesses are of the same size. For both cases, the number of weight accesses equals filter size (KKD) times the number of filters (N).

We only need $2M$ input accesses for event-based processing - one write and read per event. However, assuming that events are processed sequentially, many outputs would need to be updated for each input, leading to M event updates of KK results for N filters.

For frame-based processing, we need M event writes, and C ROI reads of size RRD on the input side. For outputs, if the convolution is completely unrolled spatially, all maxima per

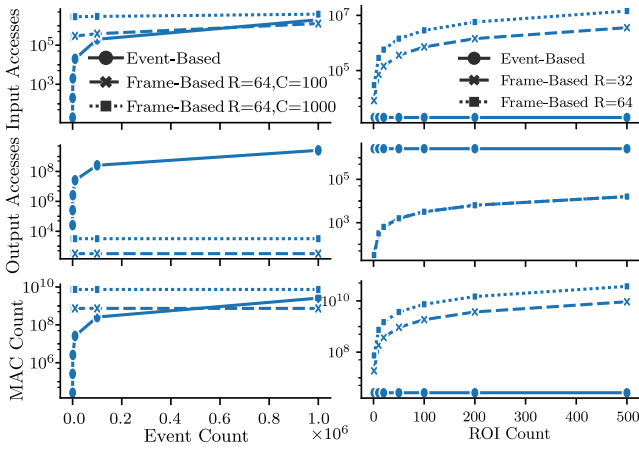


Fig. 3. Input (top) and output (middle) memory accesses along with MAC count (bottom) for varying event counts (left) and ROI counts (right).

ROI can be determined at the same time, requiring N output writes per ROI (CN total).

We measure the required computation in multiply-accumulate (MAC) operations. We convolve M events with N filters of size $K \times K$ for event-based processing. For frame-based, C ROIs of size $R \times R \times D$ are convolved with N filters of size $K \times K \times D$. Note that this does not account for optimizations that may be possible in some specific cases.

We assume a $R = 64$ ROI size, with $D = 7$ time step channels. Filters are $N = 32$, $K = 9$, and $D = 7$ pregenerated spatiotemporal filters. These parameters were chosen in accordance with Section II-C.

Fig. 3 (left) shows how memory access and MAC counts vary with event count for event-based processing and frame-based with 100 and 1000 ROIs. At a low-event count, event-based processing requires significantly fewer input accesses and less computation, but that advantage is quickly lost when the event count grows. At the 500k event mark, event-based compute and input accesses are comparable to a frame-based version with 100 ROIs. More importantly, event-based processing requires significantly more output accesses than frame-based processing.

We then compared the results for a fixed event count ($M = 1000$) but a varying ROI count and two different sizes ($R = 64$ and $R = 32$). Results are shown in Fig. 3 (right). If the number of ROIs is kept low, frame-based processing does not significantly increase the computation.

As this analysis shows that the benefits of using an event-based method would be uncertain, we have chosen to use frame-based processing for the following reasons. We can use established CV algorithms. We have substantially smaller output memory requirements due to ROI use and the option to max pool the outputs (discussed in Section III-C). We can reduce computations due to input noise with ET (also discussed in Section III-C). Finally, the structured frame data is much more favorable to parallel processing.

C. Example Tracking Pipeline

While the goal of this work is not to drive algorithmic improvements, it is necessary to demonstrate that the computational model described above is indeed representative

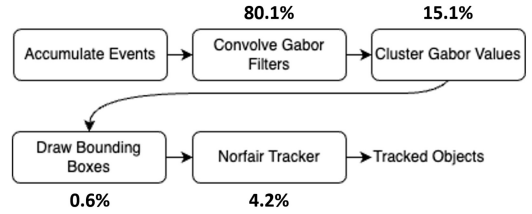


Fig. 4. Object tracking pipeline with the percentage of GPU runtime marked. Note that we did not include event accumulation in the total runtime.

of a tracking application, as the insights from this model drive our architecture design. To do that, we implemented a complete filter-based object-tracking pipeline in Python using well-established and widely used algorithms. Spatiotemporal Gabor filters have been used for tracking in several prior works [25], [26]. They consist of sinusoids multiplied by a Gaussian function. In our case, we use a 3-D Gabor filter where different rotation angles correspond to different kinds of motion. A block diagram of the pipeline is in Fig. 4. We first accumulate events into frames given an accumulation interval. We split these frames into 64×64 pixel ROIs. We choose ROIs by selecting all regions containing existing tracked objects while taking the estimated velocities of those objects into account to include the estimated future location of the objects. Every 30 frames we process the full frame to detect new objects. We use 3-D filters with 8 different rotations in the x - y plane corresponding to the orientation of the direction of motion and 4 different tilts in the time dimension that correspond to different speeds for a total of 32 different filters. Our filters have seven time channels, 2 ms each, so we use the most recent 14-ms period to construct each ROI. Our filters are $9 \times 9 \times 7$. To eliminate inaccuracies due to zero-padding when processing adjacent ROIs, we consider the center 56×56 pixels of the output valid and overlap the ROIs by 8 pixels for full coverage. We then convolve the filters over them. The resulting output values for each filter are thresholded and then clustered using the DBSCAN algorithm [27]. We generate bounding boxes for each cluster and fuse the bounding boxes into a single set of boxes per frame, using the Weighted Boxes Fusion algorithm [28]. We pass these fused bounding boxes to the Norfair tracking algorithm. We use well-established algorithms since we do not want to tie down our architecture to insufficiently proven methods. We also measured the execution time of each part of the pipeline using a CPU (Intel Xeon E5-2695) and a GPU (NVIDIA GeForce RTX 2080 Ti, 27 TOPS peak performance). As we are considering a stationary-sensor application, this likely more compute than would be available for mobile, embedded, or near-sensor processing, but the general conclusions should be broadly applicable.

As shown in Fig. 4, most of the runtime is consumed by the convolution of the filters, so accelerating this stage of the pipeline will have the most impact on the overall tracking speed. Furthermore, the clustering step, which utilizes DBSCAN, has $O(n^2)$ complexity. Using our convolution-based filtering beforehand reduces the number of candidate points by a factor of 6 in the *Birds* dataset described below. This reduction gives a $36 \times$ speedup compared to simply clustering the raw events. Our pipeline achieved 10 fps (12

TABLE II
TRACKING ACCURACY RESULTS OF THE PIPELINE ON THE BIRDS
DATASET WITH VARIOUS FILTER PRECISION

Precision	MOTA	IDF1	HOTA
Floating Point	47.8	72.4	51.7
16-bit Quantized	47.8	72.4	51.7
8-bit Quantized	47.9	72.2	51.4
6-bit Quantized	47.3	72.1	51.3
4-bit Quantized	7.2	52.9	40.3

TABLE III
TRACKING ACCURACY RESULTS OF THE PIPELINE ON THE BIRDS
DATASET WITH VARIOUS EFFECTIVE FRAME RATES. LOWER-FRAME
RATES SKIP TIME STEPS IN THE EVENT DATA

Frame rate (fps)	MOTA	IDF1	HOTA
500	47.3	72.1	51.3
250	7.1	55.5	41.6
166	10.0	57.0	41.0
62	-46.9	20.1	18.7

for convolutions only) on the CPU and 36 fps (53 fps for convolutions only) on the GPU.

It is also worth noting that many other applications well suited to event cameras also use a convolutional kernel and could be accelerated using our accelerator or one with some modifications. Some of these possible applications include particle size monitoring, high-speed counting, edgelet tracking, and optical flow [29], [30], [31], [32], [33].

To show that event-based data does not require high precision for processing, we compare the relative tracking accuracy when using floating-point and varying bitwidth integer filter coefficients in the pipeline. We evaluated this using an in-house *Birds* dataset collected using the DVXplorer camera. This dataset shows birds flying from right to left across the screen. The resolution of the camera is 640×480 pixels. All inputs are ternary, -1 (decrease), 0 (no change), and $+1$ (increase). For object tracking, we used metrics multiobject tracking accuracy (MOTA), identification F1 score (IDF1), and higher-order tracking accuracy (HOTA) described in [34] to evaluate our pipeline using standardized metrics from the multiobject-tracking (MOT) Challenge. We observed that the accuracy loss was negligible down to 6-bit integer weight precision, after which it drops off sharply, as can be seen in Table II, confirming that convolving event-based data with Gabor filters does not require high-numerical precision. SC is capable of higher precision, but we did not need it for our application. In prior works, SC has been used successfully with up to 8-bit precision [35], [36].

We have tested the accuracy of the pipeline using varying effective frame rates by skipping events from certain time steps. Results are in Table III. It shows that for event-based data it is imperative to maintain high-processing throughput, on the order of at least hundreds of fps, which is beyond what a CPU or a GPU can handle (as per our results above), necessitating a custom hardware approach.

To justify our choice of the algorithmic approach, we compared the 6-bit integer Gabor filter pipeline with the tracker described in [37], which uses a dataset collected using a DAVIS camera with a telescope set up to view moving space junk. We refer to this dataset as *Space Junk*. The pipeline

TABLE IV
GABOR PIPELINE ACCURACY ON SPACE JUNK DATASET
COMPARED TO [37]

Pipeline	Sensitivity	Specificity	Informedness
Gabor Filter (4-bit int)	0.796	0.996	0.792
Gabor Filter (6-bit int)	0.817	0.998	0.815
Gabor Filter (8-bit int)	0.820	0.996	0.816
Gabor Filter (16-bit int)	0.823	0.996	0.820
Feature Detector + Tracker	0.782	0.992	0.775

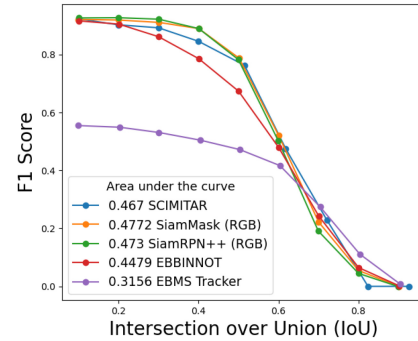


Fig. 5. Comparison on EBBINNOT *Cars* Dataset. Additional comparisons beyond SCIMITAR and EBBINNOT are from [38].

used in this article is significantly more computationally intensive than ours since it uses an exponentially decaying time surface to represent the *history* of a given pixel rather than accumulating events into discrete frames. We used a subset of the data (by discarding complete videos with no objects) and used the same accuracy metrics as described in [37]. The abbreviations are as follows: true positive (TP), true negative (TN), false positive (FP), and false negative (FN)

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad \text{and} \quad \text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \quad (1)$$

$$\text{Informedness} = \text{Sensitivity} + \text{Specificity} - 1. \quad (2)$$

As shown in Table IV our tracking approach achieves slightly higher accuracy than the feature detector + tracker method in prior work. We also observe no significant gain in accuracy by increasing to higher than 6-bit precision. Given the straightforward computational nature of filter-based computing (3-D convolutions), we build our architecture around them. It also gives us the flexibility to switch to different types of filters or tackle other applications based on 2-D or 3-D filters.

To validate with larger objects, we compared to the car dataset in [38]. We will refer to this dataset as *Cars*. In Fig. 5 added our results to a comparison from [38] of detection F1 scores for different detectors for different intersection over union identification thresholds. See [38] for a description of the additional traces.

D. Stochastic Computing

Given the above evaluation, for our architecture, we are looking for technologies that enable low-precision convolutions in a fast, efficient, and high-compute-density manner to sustain a high-effective frame rate. We leverage two techniques that offer unparalleled compute density: 1) SC [6], [39] and 2) CIM [11]. In this section, we will introduce the first of the two techniques.

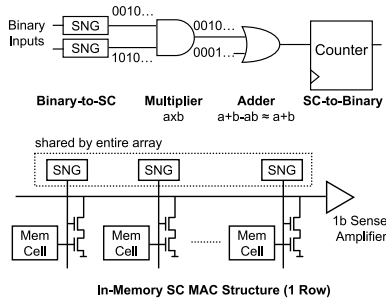


Fig. 6. (Top) Basic concept of SC. Note that the AND gate acts as a multiplier and the OR gate acts as an approximate adder that is accurate when the product ab is small. (Bottom) In-memory SC MAC structure. This figure shows an implementation of an AND between input stored in the memory cell and the weight stream generated by the SNG followed by an OR to add with the other products in the MAC.

A comprehensive overview of SC is presented in [39]. Fig. 6 shows the basic concept of SC. SC represents numbers as the fraction of 1s in a bit stream instead of as binary. A SNG converts binary to stochastic numbers. SC has been extensively explored for uses in CV applications [40], [41], [42] and machine learning [6], [7], [8], [10]. Our application fits the strengths and avoids the typical drawbacks of SC described below.

- 1) *Single Logic Gate Operations*: Multiplication arithmetic uses an AND gate to perform a bit-wise AND and addition is implemented using an OR gate to perform a bit-wise OR. A simple counter can convert stochastic to binary numbers. This compact hardware unit enables many large, dense MACs in a small area with high-spatial reuse [6], [9].
- 2) *Variable Precision Support*: Each cycle of SC approximates the result. Changing compute precision on the fly through ET to take advantage of sparsity reduces the computation time.
- 3) *Accurate Multiplications*: The stochastic nature of SC causes random errors [39]. Since the inputs for our application are $+1$, -1 , and 0 , one full stream is either 1 or 0. Due to our choice of pseudorandom number sequences (as in [35]), multiplications are guaranteed to be accurate if they run for the full stream length.
- 4) *Low-Precision Requirements*: Increasing the precision of SC requires an exponential increase in stream length (limiting SC to applications requiring eight or fewer bits of precision), but that is not a problem here as we have low-precision requirements (1-bit inputs and 6-bit weights). Prior works have shown SC can match fixed-point precision in this range efficiently [35].
- 5) *Amortized Conversion Cost*: SC requires costly conversion from binary to stochastic streams [6], [7], but the parallelism in our application allows us to amortize the conversion costs of filter weights (32-way reuse in our architecture) while inputs, being binarized, have no stochastic number generation overhead.

III. SCIMITAR IMPLEMENTATION

Based on the algorithm-driven analysis from Section II-B, our architecture must support low-precision, sparse 3-D convolutions at high throughput and energy efficiency. Our hardware

requirement based on our tracker is $32 \times 9 \times 9$ filters with up to eight time channels (7 required for our pipeline), a 64×64 ROI, 6-bit filter precision, and 10-bit output precision.

A. Stochastic Computing In-Memory Macro With In-Situ SNG

We use SCIM to achieve high-energy efficiency for processing the sparse reconstructed event-camera ROIs. Embedding SC computations in memory has previously been shown to be an efficient way to implement the AND-OR structure of the large MAC used in the convolution [9], [10], [43]. Fig. 6 shows the structure of an SC MAC unit in an SRAM-based SCIM macro. Each SRAM cell stores a stochastic bit of the weight parameters. We add two transistors next to the cell to perform an AND operation (multiplication) between the stored bit and the compute word line. We describe our improvements on this below.

The main challenge of embedding SC in memory is converting a binary number to a stochastic bit sequence in memory. Previous works have stored preconverted unrolled streams in memory, but this method requires significant area and hurts the overall energy efficiency [10]. Our recent work uses a simple and compact solution to embed the binary-to-stochastic conversion in memory [44]. Fig. 7 shows an overview of the SCIM macro. The core of the macro is an array of SCIM units, which are the basic processing units corresponding to one weight parameter each. The SCIM *unit* consists of memory cells for storing one weight in binary format, an in-situ SNG for number conversion, and multiple SC MAC units reusing the SNG output. We chose 6 bits as the precision for weights since the analysis presented previously indicated that it is sufficient for our target application. The in-situ SNGs are embedded next to the weight storage cells to convert them from binary to stochastic bit sequences. An extremely high-weight reuse factor is achieved by sharing the in-situ conversion output with many in-memory SC MAC units, amortizing the cost of the in-situ conversion. We embed 32 SC MAC units per weight to maximize weight reuse since this is the largest number of MAC units that can fit within the memory cell's height. An SCIM slice consists of 81 SCIM units and corresponds to an unrolled 9×9 filter for one of the eight time channels. We process the left and the right of the ROI rows sequentially due to the physical limitations of the macro layout. 32 SCIM slices within the macro share the same inputs, each implementing a different filter.

The SNG is embedded inside the memory to achieve in-situ number conversion. During computation, the in-situ SNGs use random numbers generated from pseudo-random number generators (PRNGs) near the macro to convert the stored binary number to stochastic bits. The PRNG uses shift registers to store the entire output sequence of an XOR-based, maximal-length linear feedback shift register (LFSR) and circularly shifts in each cycle. We choose the LFSR's polynomial order to match the bit width of the weight coefficient, which guarantees accurate conversion since the LFSR outputs are uniformly distributed over the stream length. Each PRNG generates unique random numbers for 32 SNGs. Three PRNGs in each macro store different LFSR sequences and can support

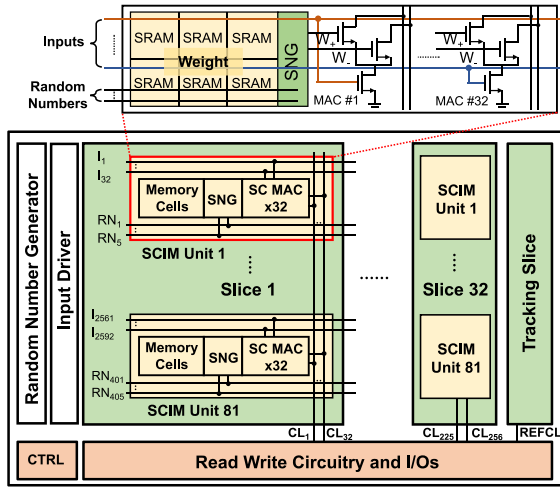


Fig. 7. SCIM unit with 32 MAC reuse and SCIM slice (top) and SCIM macro architecture (bottom).

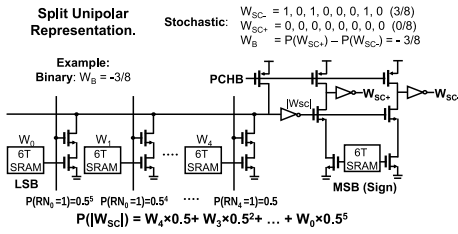


Fig. 8. Split-unipolar stochastic number representation (top) and in-situ SNG circuit (bottom).

81 SNGs without correlation. Note that we use the unrolled sequence for the PRNG since the cost is shared and to allow testing of non-LFSR sequences. Since the correlation is limited to the stochastic bits computed within each MAC unit at each cycle, the minimal number of unique random numbers should match the dot product's size (81). We can reuse this sequence for all 32 slices within the macro. Further, given convolution's extensive data reuse for both weights and inputs, conversion costs, as well as SNG cost itself, are heavily amortized, as has been shown in prior works [6].

The in-situ SNG is shown in Fig. 8. For a 6-bit weight, the fifth bit is selected by random numbers with a probability of 0.5, the fourth bit is selected with a probability of 0.25, and the LSB is selected with a probability of 0.5^5 . Each bit cell has two extra cascaded nMOS transistors beside the 6T SRAM cell to perform an AND operation between the stored binary bit and the random number. The output of AND logic in each cell is connected to form a local bitline, which performs a wired-OR operation. An inverter amplifies in-situ SNG's local bit line and inverts the signal to maintain the correct logic. Using the in-situ SNG also makes the implementation agnostic to the stream length, which was not the case for bit-parallel SCIM [10]. A more detailed presentation of the in-situ SNG macro is in [44].

For event-based object tracking we require supporting signed inputs (event polarity) and weights (filter coefficients). We use a split-unipolar representation to support signed numbers [6]. A number is represented by two SC streams: 1) W_{SC+} and 2) W_{SC-} only one of which is enabled at a time by the sign bit. The value of the number is encoded as the difference between them: $W_{SC+} - W_{SC-}$. If the number

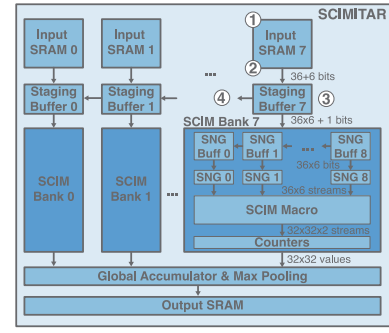


Fig. 9. SCIMITAR architecture block diagram.

is positive, W_{SC+} represents the value's amplitude, and W_{SC-} is a zero stream, and vice versa for the negative numbers. A demultiplexer circuit using pseudo-nMOS logic generates the split-unipolar streams with an inverter buffer output.

Output values are 10 bits. This is due to 6-bit weights, eight time channels (+3 bit), and split unipolar (+1 bit). A split unipolar SC MAC can have activity on both polarities within a stream).

B. SCIM Accelerator Architecture

Given the algorithm and circuit-driven design of the SCIM macro, we now face the challenge of integrating it into an accelerator architecture that can also take advantage of vast event-data input sparsity. Fig. 9 shows the overall architecture of SCIMITAR. Given a limited set of operations, we implement the control logic as a finite-state machine (FSM) controlled through a set of programmable registers. The *I/O interface* transfers ROIs to input SRAMs and outputs/maxima from the output SRAM. We designed SCIMITAR to process a single 64×64 ROI at a time with up to eight time channels. Based on that, we organized the architecture into eight *columns*, each consisting of an *input SRAM*, *staging buffer*, and a *SCIM bank*. We provisioned each input SRAM to double buffer one time channel of a single ROI. Input SRAM width is provisioned to hold 64 2-bit (ternary) values.

Values from input SRAM are first read into staging buffers, then optionally rotated, and passed onto the SCIM banks. Since, as described in the previous Section, the SCIM macro can only process half of the row at a time, staging buffers are provisioned for 36 2-bit values. Within each SCIM bank, we write the input values to the SNG buffers where they are used to generate SC streams when the computation starts. Each bank has nine SNG buffers, which hold nine rows of 36 values, making it possible to unroll one 9×9 time channel of the convolutional filters spatially. This spatially unrolled convolutional window is based on the one used in [6] to maximize convolutional data reuse.

The weights are preloaded in the SCIM, and their streams are generated in situ, as described in the previous Section. Within each macro, a sliding 9×9 convolution is performed across nine input rows, generating 32 outputs for each of 32 filters for a total of 1024 outputs per bank. Outputs of each compute line are fed into counters. After the computation finishes, counter outputs are sent to the global accumulator block. SCIMITAR adds outputs of eight counters to implement the combined $9 \times 9 \times 8$ filter size.

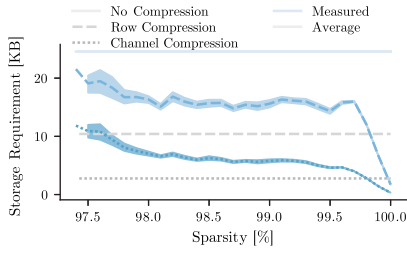


Fig. 10. ROI memory requirements for different compression schemes.

We will now describe the architectural optimizations indicated in Fig. 9. Those optimizations improve the efficiency of sparse event-data processing (algorithm-driven) and circumvent the limitations of SCIM-based computing (circuit-driven).

1) *Channel Load Skipping*: While ROI processing reduces memory and computation compared to full frames, individual ROIs are also highly sparse. To take advantage of this sparsity, we propose embedding additional information in input memory to avoid storing and loading rows of the ROI with no events. Since data in memory is organized in rows, we consider two levels of granularity: 1) *row* and 2) *channel* skipping. The former will skip any slice of 64×8 pixels (one row across all eight time channels) that is all zero. The latter will skip any slice of 64 pixels (one row, one time channel) that is all zero.

To support this functionality, input memory contains the *next_row_id*, which indicates the index of the next nonzero row stored in the subsequent address. In case of row skipping, the next row index is shared across all eight input SRAMs. For channel skipping, each SRAM has the index information for the next row. To evaluate potential storage compression, we used *Birds* dataset described in Section IV, partitioned it into $64 \times 64 \times 8$ ROIs, and calculated the memory required for each of the ROIs, including *next_row_id* information. Results are shown in Fig. 10. Row skipping reduces storage requirements by $2.36\times$ on average, while channel skipping does so by $8.88\times$ on average, even including indexing overheads. Given those results, we opt to implement channel skipping in SCIMITAR.

We implemented channel skipping in local control logic on a column-by-column basis. Whenever reading a word from input memory, if *next_row_id* is more than *current_row* + 1, where *current_row* is the index of the currently read row, local control logic skips the next N reads, where N is equal to the $next_row_id - current_row$. This is shown schematically in Fig. 11. In other words, local input SRAM control will wait until global FSM catches up to its next nonzero row. We always read the first row in an ROI since there is no *next_row_id*.

2) *Half-Row Multiplexing*: The SCIM Macro can only process half of the 64-wide row at a given time. Given the relatively high-energy cost of accessing SRAM, it is prudent to store half-rows separately in memory to save on accesses. However, to avoid a gap in convolution coverage, each half needs to include the same 8-pixel overlap region. Further, *next_row_id* information would need to be stored with each half-row. Instead of storing $64 + 6 = 70$ bits, we would need to store $2 \times (36 + 6) = 84$ bits, a 20% storage overhead. Instead, we partition each input SRAM into three physical

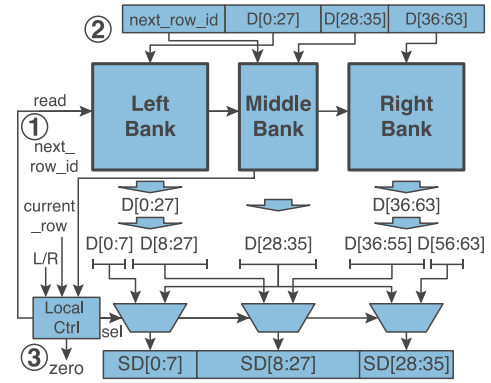


Fig. 11. Channel load skipping and half-row multiplexing using partitioned input SRAM. Each ROI row is split into three parts (left, middle, and right), placed in their respective SRAM banks. Only two banks are accessed at a time, depending on which half is being processed (L/R control signal). *next_row_id* is stored in the middle bank, which is always accessed (overlap region), and is compared against *current_row* to skip rows containing all zeros.

banks: 1) left (bits 0–27); 2) middle (bits 28–35); and 3) right (bits 36–63), as shown in Fig. 11. A signal from the control FSM (L/R - left/right) decides which banks are accessed (left-middle or middle-right) and multiplexes the outputs to appropriate positions of the staged data *SD*. This approach avoids storage overheads while saving access energy. *next_row_id* is stored in the middle bank, as it is always accessed.

3) *Zero Indicator*: Staging buffers also contain a zero indicator bit. Upon detecting one or more zero rows, using the *next_row_id*, local control will also set the zero indicator bit in its staging buffer, as shown in Fig. 11. This bit is used downstream to gate the SNG buffer propagation and the SNG stream generation for that row. This approach saves energy by reducing the toggling of unnecessary logic.

4) *Time Channel Overlap*: SCIMITAR supports up to 8 time channels in each ROI. In some applications, subsequent “frames” can be disjoint, meaning their time channels cover nonoverlapping time windows. For example, if using 1-ms channels, the first reconstructed frame covers the first 8 ms, the second one the next 8 ms, etc. However, temporal resolution can be improved if there is an overlap between subsequent “frames.” For example, each subsequent reconstruction can shift by 1 ms, where the remaining 7-ms overlap. Given that in SCIMITAR, filter time channels are assigned to physical SCIM macros, each of which is connected to its input SRAM, naively supporting such overlap would require reloading the entire ROI, as time channels would need to be physically moved between input banks. To provide seamless support for overlapping time-channel ROIs, we propose to connect staging buffers as a circular buffer, as shown in Fig. 12.

Initially, time channels are properly aligned to columns, for the first 8 time steps. Half-rows can be loaded directly into staging buffers and passed to their respective SCIM banks. After processing, time channel $t = 0$ is replaced with time channel $t = 8$ in column 0. After loading each row to the staging buffers, they are rotated once, so that channel $t = 8$ ends in column 7, channel $t = 7$ in column 6, and so on. Using this approach, time channels can be overlapped with a minimum number of memory accesses. The latency of rotating staging buffers is hidden using SC stream processing latency. In the worst-case scenario, SCIMITAR needs to hide 7 rotation cycles.

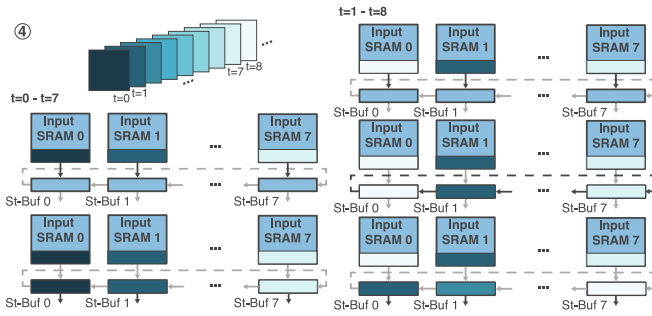


Fig. 12. Time channel overlap using circular buffers.

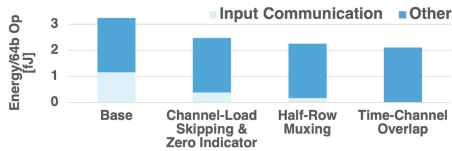


Fig. 13. Impact of proposed optimizations on the computational energy efficiency of the SCIMITAR architecture. Efficiency calculated on 99% sparse input data assuming 32×2 -bit SC streams (64 cycles).

Fig. 13 shows how energy per computation, for a 64-long SC stream MAC, is affected by the above optimizations on the accelerator level. Our use of channel-load skipping, half-row multiplexing, and time-channel overlap, can reduce the energy related to input communication (SRAM, buffers) by up to 55 times, which translates to $1.54\times$ higher-energy efficiency for the entire accelerator.

SCIMITAR can scale well in terms of sensor size and throughput. Since processing is ROI-based, any sensor size can be supported, subject to throughput constraints. Additional arrays can be added to process more ROIs in parallel, improving throughput. Temporal resolution is adjusted through the number of SCIM arrays per filter. Certain aspects of the architecture, like the ROI or filter size, are tightly coupled to circuit design. However, we argue that tightly coupling microarchitecture and circuit design allows us to achieve significant performance improvements. Larger ROIs can be supported by “stitching” multiple smaller ones. SC also enables flexibility in computation precision [6] - by adjusting the stream length we can tradeoff precision with latency and energy. We leave exploring the interaction of variable precision with tracking accuracy to future work. Here, we use equivalent 6-bit precision streams, which guarantees good tracking results, as demonstrated in Section II-C.

Since SCIMITAR is built for 3-D convolutions, it is similar to prior works that use SC for convolutional neural networks (CNNs) [6], [10]. However, there are differences. SCIMITAR is built for shallow filters, while the ones used in CNNs have tens or hundreds of channels [45]. Further, SCIMITAR has a host of optimizations for exploiting very high-input sparsity, uncommon in CNNs. However, concepts like the in-situ SNG could be used in SCIM accelerators for ML, which we will explore in future work.

C. Early Termination & Maxima Tracking

Every cycle of SC computation provides an estimate of the final result. For example, if the first 16 cycles of computation contain 50% ones, we can expect that at the end of the

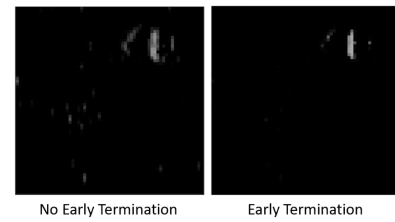


Fig. 14. ET noise reduction. ET helps eliminate medium to low-valued peaks, which effectively denoises the output.

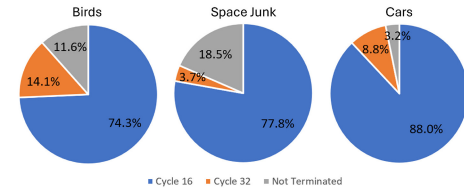


Fig. 15. Percentages of ET at cycles 16 and 32. Computed over subsets of ROIs from *Birds*, *Space Junk*, and *Cars*. Note that ET thresholds are dataset specific.

computation, the proportion should be similar [7]. Short streams can give an estimate, and longer streams increase the precision and reduce the impact of randomness. This progressive convergence to the final result leads to the concept of ET. Since early partial results from running a computation approximate the final result, we can judge whether or not we are likely to care about the result of a computation before it is complete. While ET has been proposed before, it was either static (termination after a fixed number of profiled cycles) [7], [46], or required complex decision methods that cannot be easily implemented on an output-by-output basis [47]. Since we are looking for peaks in our application, we can rule out negative values and values close to 0 after a certain number of cycles. This saves power and latency especially when processing inputs that only contain noise and thus do not produce peaks above the threshold. Note that ET can be used to dampen noise if the threshold is set that will terminate the values expected from white noise early. As shown in Fig. 14 real peaks are unaffected.

The downside of using partial results is that the shorter the stream is, the more likely it is to have a large error. An error of 1 on a 16-bit stream is $1/16$ whereas an off-by-1 error for a 4-bit stream is $1/4$, which is significantly more serious. To reduce the risk of such errors affecting accuracy, we do not use ET on the first 16 cycles of computation and only turn ET on after that. Since we split the signed compute streams temporally for inputs, the first 8 bits of signed compute take 16 cycles, 8+ and 8-. After that, we periodically check the result of the stream to see if it is below a threshold that would allow us to discard the pixel. In an ideal case for ET, we could check the count against a threshold every cycle, but in our case, we are constrained to check after we have computed an equal number of positive and negative cycles. For a 32-bit stream with 64 cycles of compute, this gives us the potential to save up to 75% of compute time, with a high degree of confidence that we will not lose any peaks that we care about, as shown in Section IV.

Constraining ET to occur only at cycle 16 or 32 simplifies the hardware and reduces the overhead of switching between

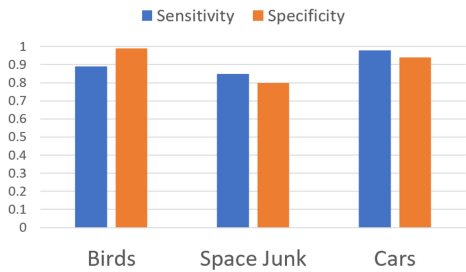


Fig. 16. Sensitivity and specificity for ET SC ROI peak matching with floating point. This is computed over a subset of ROIs from each dataset with ET levels in Fig. 15.

positive and negative compute streams. Since the relevant ET threshold depends on the stream length, only checking at the power of 2 stream length values allowed us to set a threshold and use a bit shift to scale the threshold from the 16th to the 32nd cycle. Fig. 15 shows that most ET happens at the earliest opportunity. While finer-grained ET is possible, the benefits we get for earlier checks for ET get progressively smaller. Terminating after 32 cycles saves about half the computation time. After only 16 cycles, terminating saves an additional 25% compared to 32, but terminating after eight cycles would only save an additional 12.5% compared to 16 in an ideal system. The benefits are small compared to the overheads of pipelining and loading. The overall reduction in compute cycles we observed over 7000 ROIs from the *Birds* dataset was 62.8%, see Fig. 15. The “*Space Junk*” and *Cars* datasets showed similar cycle reduction. Sensitivity and specificity for peak matching with ET are shown in Fig. 16.

Once we compute all the outputs we need to find a maximum for each filter. However, storing all intermediate results takes a large amount of memory. For 32 filters, an ROI of 64×64 pixels, and 10 bits per output, we need to store $64 \times 64 \times 32 \times 10b = 1.28$ MB per ROI. To reduce memory size, we do not want to store all the outputs. A single max will do well if we only have a single object in an ROI, but multiple objects will fail with a single max, see Fig. 17. One way to solve this issue is to save the maximum value from each cell in a grid of regions in the output. This method tracks additional outputs. We instead decided to keep track of a per-ROI-column maximum on the fly as shown in Fig. 18 (left). This option is straightforward to implement in hardware, as each column is processed sequentially in the same output counter, requiring only two additional registers per counter: 1) value (10 bits) and 2) row index (6 bits) of the current maximum. Beyond the hardware simplicity, we selected column max instead of a grid option as it showed better-peak retention than the grid option using equivalent storage as shown in Fig. 18 (right). It reduces the output memory requirement to only $64 \times 32 \times (10 + 6)$ bits = 32 KB per ROI.

IV. EVALUATION

A. Object Detection and Tracking

To evaluate the SCIMITAR architecture, we ran RTL emulation using a scaled-down version of our SC architecture on the ALVEO U200 FPGA implemented fully in digital logic. We used functionally equivalent digital logic to replace the CIM for the test on the FPGA. The mixed-signal verification of the

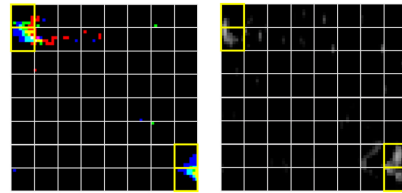


Fig. 17. Example ROI with multiple objects. The left image shows a few time channels of the input. The right image shows the result after convolving with a Gabor filter.

	Storage	Peaks	Total Bits	Normalized Storage
All	N/A		32,768	1.0
16x16 Grid	68	3,072		0.094
Column Max	68	896		0.027
8x8 Grid	60	896		0.027
4x4 Grid	58	256		0.0078
2x2 Grid	57	72		0.0022
1 Maximum	54	20		0.00061

	Birds			Space Junk			Cars	
	HOTA	MOTA	IDF1	Sensitivity	Specificity	Informedness	F1/IoU Curve Area	
No Column Max	51.3	47.3	72.1	0.817	0.998	0.792	0.467	
Column Max	42.5	27.1	61.6	0.679	0.998	0.677	0.458	

Fig. 18. Column max. In column max, we save one maximum from each output column. This technique substantially reduces storage requirements while catching peaks when multiple objects are in the ROI. In the table on the right, we look at different grid options compared to column max for a Gabor filter. Column max has the same storage requirement as an 8×8 grid while detecting the same peaks detected by a 16×16 grid and requiring less additional hardware than the grid options. Tracking results are still reasonable after applying column max as shown in the lower half of this figure in the chosen metrics for each dataset.

SCIM macro is described in the next section. We evaluated 100 consecutive reconstructed frames from the *Birds* dataset for 7000 total ROIs. We use the same filters as in Section II-C, converted to 6-bit integer values by scaling maximum filter values to 31 and then quantizing them. We do not train or modify filter coefficients to better suit our SC computation in this work. This means existing filter coefficients can be used directly without modifications other than quantization to 6 bits. We made no additional modifications to the algorithm to suit our hardware. By doing this, we verified three points: 1) close match between peak locations using SC and floating point; 2) little to no loss of peaks due to column max; and 3) little to no loss of objects due to ET. We used the sensitivity and specificity metrics described earlier treating the peaks from floating-point convolution results as ground truth and comparing them to the peaks simulated results using SC. Sensitivity, specificity, and informedness are 1 for perfect identification and 0 for entirely incorrect identification. As shown in Fig. 16, the sensitivity is above 85% and the specificity is above 95%.

We set the identification thresholds for floating point and SC so they produce a similar number of peaks to reduce the dependency of the result on arbitrary thresholds. This means we have nearly equal FP and FN rates for the shown results. Here, a correct identification consists of the same ROI being identified as containing an object by both floating point and SC. A FP is an ROI identified by SC but not floating point and a FN is an ROI identified by floating point but not SC. For the stream length of 32 bits or 64 cycles, the average pixel error in the location of the peak identification was 3.36 pixels.

Column max tracking did not result in any loss of peak identification accuracy compared to global maximum tracking.

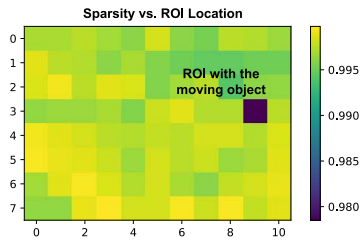


Fig. 19. Sparsity versus ROI location of one frame. The ROI with the object has a sparsity of 98.5% compared to the other ROIs with sparsities above 99%.

However, the *Birds* dataset always has a relatively small number of objects in the ROI. The chances of two objects being located above each other so that all peaks from one of the objects are blocked by the other are low. Also, there was no loss of accuracy due to ET since we chose a threshold well below the threshold used to identify peaks. More aggressive ET thresholds could further improve performance but might start to impact accuracy.

B. Hardware Evaluation

The SCIM macro is custom-designed, laid out, and simulated in GF 12 nm LP technology using the layout-extracted netlist in Cadence Virtuoso’s analog design environment. We verified the functionality of the read/write and computation operations with input vectors that lead to worst-case timing conditions. The timing and energy of the macro are characterized in different PVT corners to generate library files for top-level digital flow. We reported the energy efficiency of the overall SCIMITAR accelerator from simulations using Cadence Genus and Synopsys PrimeTime. Since sparsity is critical for the system energy efficiency, the event camera’s data is studied as shown in Fig. 19. When a moving object is flying in front of the event camera, the ROI containing the object has a sparsity of 98%, while the other ROIs have a sparsity above 99%. To evaluate the system’s energy efficiency at different sparsity levels, we generate randomly distributed images with sparsity varying between 0 and 99%. The results are also cross-validated by getting the energy consumption of each block factoring in its expected switching activities. For the 0% sparsity case, we assume each input has a switching activity factor of 0.5. We pessimistically assume dense outputs that switch 50% of the time. The SCIM macro runs computation for 64 cycles to generate a full-precision output, consuming the most energy. The energy breakdown of the 0%-sparsity case is shown in Fig. 20 (bottom left). The total energy consumption is 11.9 fJ or 83 TOPS/W.

SCIMITAR’s energy efficiency improves with sparsity, a notable advantage over other CIM solutions. SC uses combinational circuitry, which naturally consumes less energy when the switching activity decreases. The energy of the SCIM macro’s input driver and in-memory MAC unit scales down proportionally with increasing sparsity levels. 32 SC MAC units within an SCIM unit share each in-situ SNG and PRNG. The SC MAC unit only consumes 1/4 of the energy in the sparse input case compared to the dense input case. However, we do not scale the PRNG or in-situ SNG’s energy with sparsity due to the constant switching activities of PRNG. Flip-flops that are active every clock cycle dominate the

Technology		GF 12nm LP	
Input / Weight Precision		Ternary / 6-bit	
(Multiply & Accumulate) Compute Performance			
Peak Throughput	No Early Termination	21 TOP/S	
	Early Termination Birds	56 TOP/S	
	Early Termination Space Junk	53 TOP/S	
	Early Termination Cars	71 TOP/S	
Energy Efficiency	No Early Termination	0% Sparse	83 TOP/S/W
		99% Sparse	482 TOP/S/W
	Early Termination	Birds 99% Sparse	1285 TOP/S/W
		Space Junk 92% Sparse	733 TOP/S/W
		Cars 97% Sparse	1702 TOP/S/W
Object Tracking System Performance			
Energy Consumption		1.9×10^9 ROI/Filter/J	
Throughput		84.4×10^6 ROI/Filter/sec	

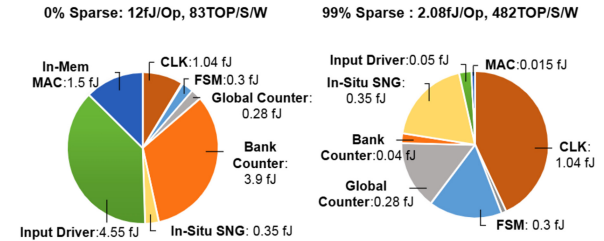


Fig. 20. SCIMITAR performance summary (top). Energy breakdowns without ET for 0% (bottom and left) and 99% (bottom and right) sparsity in fJ and normalized by the number of operations.

bank output counter’s energy consumption. The input-based clock gating can significantly reduce the counter’s energy. The remaining parts that do not scale with the sparsity are clocking and state machines. The energy breakdown is shown in Fig. 20 (bottom right). The energy efficiency for the 99%-sparse input is 482 TOPS/W. We calculate sparse energy efficiency factoring in the “skipped” computation, similarly to other sparse accelerators [48], [49], [50].

ET can stop a computation before it reaches the end of the SC sequence length at 64 cycles, saving both time and energy. If the computation terminates, all the SCIM banks will stop and move to the next ROI. Since we skip the remaining computing cycles, the energy consumption and throughput improve by the percentage of skipped cycles. The simulation results show a $2.7\times$ improvement in energy efficiency and peak throughput. When we enable ET for the *Birds* data set, energy efficiency improves from 482 to 1285 TOPS/W and peak throughput increases from 21 to 56 TOP/S. With ET turned on, the object tracking system achieves performance of 1.9×10^9 ROI/Filter/J and 84.4×10^6 ROI/Filter/s. Assuming 32 filters and 100 ROI/frame translates to an equivalent frame rate of 26k fps, orders of magnitude higher than the rate achievable with conventional cameras [1]. It is also much higher than the 500 fps requirement from our previous evaluation, meaning that SCIMITAR could work in situations requiring processing more ROIs to avoid losing tracking performance, such as larger frames, more objects, or poor ROI identification.

C. Comparison With Prior Work

Stochastic CIM (SCIM) storing unrolled bit stream can achieve similar energy efficiency and sparsity scaling as SCIMITAR [10] but has much lower throughput per area. Since it requires 2^N cells to store an N -bit number, it has an area penalty of $2^N/N$ compared to SCIMITAR. CIM based on analog computing is widely studied for accelerating matrix

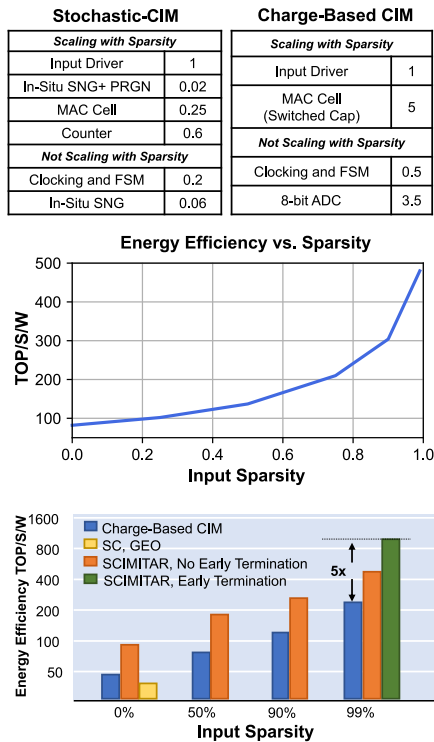


Fig. 21. (Top) Normalized energy of different blocks in stochastic-CIM and charge-based CIM macro scaled with the number of operations; and (Middle) SCIM system energy efficiency versus input sparsity without ET's help. (Bottom) Comparison of energy efficiency between SCIMITAR with or without ET and standard digital SC or ADC-based CIM (right).

multiplication in deep CNNs [11], [51]. However, few of these works focus on the object tracking application or event-based cameras [23], [52] given the relative novelty of this field. As such, no direct comparison point exists for our chosen application on CIM-style architectures or other custom accelerators. We compare to prior art based on the peak performance of ML accelerators. Note that SCIMITAR is more application-specific than a general NPU due to features, such as column max and ET, but the same architecture principles we show could be adapted to other applications or a more general purpose NPU.

Conventional CIM solutions accumulate analog current or charges on the memory's bit line and use analog-to-digital converters (ADCs) to convert analog signals to digital bits. ADCs are power-hungry and occupy a large area. Although these solutions have shown significant improvement over non-CIM digital accelerators on dense inputs, the constant energy of ADCs prevents further improvement on highly sparse inputs. Fig. 21 (top) shows the normalized energy of different components of SCIM and charge-based CIM macros scaled with the number of operations. Most components in SCIM macros are combinational circuits whose energy consumption can decrease with increasing input sparsity. The conventional charge-based CIM requires ADC to convert analog voltage to digital bits. The ADC requires a digital-to-analog converter (DAC) to generate a reference voltage and compare it with the input in each cycle, which does not change based on the input. The SCIM's system energy efficiency as a function of input sparsity is shown in Fig. 21 (middle) and a comparison of energy efficiency of SCIM, charge-based CIM, and standard

SC is shown in Fig. 21 (bottom). The CIM numbers are based on [11], and we estimate the sparsity impact by scaling the energy drawn from the input driver and switched capacitor with the input sparsity level. We also compare SCIMITAR to purely digital SC accelerator GEO [35] to show how much efficiency can be improved by combining digital SC with CIM. For a fair comparison, we compare peak energy efficiency, which is application agnostic, at 6-bit, or equivalent for SC, precision. The energy efficiency of the SCIM scales with the sparsity level from 0 to 99%, while the ADC-based CIM solution only shows negligible improvement when the sparsity level increases above 50%. Another work has shown an efficient CIM accelerator for denoising and region proposal applications for event-based cameras [23]. It performs computation between pixels but does not allow convolution and filtering by weight kernels, which is necessary for object-tracking applications.

CIM using nonvolatile memory, such as RRAM [53] and MRAM [54], [55], is less efficient than the switch-capacitor-based analog CIM [11] due to the limitation of the device. Since the RRAM and MRAM devices have large variations, the in-memory dot-product size is limited to <32 , while the switched-cap-based CIM can reach 1024. The proposed SCIMITAR is a digital CIM solution that is more efficient.

V. CONCLUSION

In this work, we proposed SCIMITAR, an accelerator for high-speed object tracking using event camera data. By taking advantage of the sparsity of the event camera data and using several techniques, including ET with SC, we achieve extremely high-frame rates. We use compute in memory to further increase compute density and energy efficiency. By achieving this energy-efficient and high-speed data-processing, SCIMITAR enables more fully utilizing event cameras for real-time tracking of extremely fast objects. To promote further work, we have made our tracking pipeline, RTL, and *Birds* dataset available for download at <https://github.com/nanocad-lab/scimitar>.

REFERENCES

- [1] G. Gallego et al., "Event-based vision: A survey," 2020, *arXiv:1904.08405*.
- [2] D. Gehrig, H. Rebecq, G. Gallego, and D. Scaramuzza, "EKLIT: Asynchronous photometric feature tracking using events and frames," *Int. J. Comput. Vis.*, vol. 128, no. 3, pp. 601–618, 2020. [Online]. Available: <https://doi.org/10.1007/s11263-019-01209-w>
- [3] P. Bardow, A. J. Davison, and S. Leutenegger, "Simultaneous optical flow and intensity estimation from an event camera," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 884–892. [Online]. Available: <http://ieeexplore.ieee.org/document/7780471/>
- [4] A. Linares-Barranco, F. Gomez-Rodriguez, V. Villanueva, L. Longinotti, and T. Delbruck, "A USB3.0 FPGA event-based filtering and tracking framework for dynamic vision sensors," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2015, pp. 2417–2420.
- [5] R. Serrano-Gotarredona et al., "CAVIAR: A 45k neuron, 5M synapse, 12G connects/s AER hardware sensory-processing-learning-actuating system for high-speed visual object recognition and tracking," *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1417–1438, Sep. 2009.
- [6] W. Romaszkan, T. Li, T. Melton, S. Pamarti, and P. Gupta, "ACOUSTIC: Accelerating convolutional neural networks through or-unipolar skipped stochastic computing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2020, pp. 768–773.
- [7] D. Wu, J. Li, R. Yin, H. Hsiao, Y. Kim, and J. S. Miguel, "UGEMM: Unary computing architecture for GEMM applications," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2020, pp. 377–390.

- [8] R. Hojbr et al., "SkippyNN: An embedded stochastic-computing accelerator for convolutional neural networks," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.
- [9] S. Li et al., "SCOPE: A stochastic computing engine for DRAM-based in-situ accelerator," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2018, pp. 696–709.
- [10] J. Yang, T. Li, W. Romaszkan, P. Gupta, and S. Pamarti, "A 65nm 8-bit all-digital stochastic-compute-in-memory deep learning processor," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, 2022, pp. 10–11. [Online]. Available: <https://ieeexplore.ieee.org/document/9980613/>
- [11] H. Jia et al., "15.1 A programmable neural-network inference accelerator based on scalable in-memory computing," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2021, pp. 236–238. [Online]. Available: <https://ieeexplore.ieee.org/document/9365788/>
- [12] T. Delbruck and M. Lang, "Robotic goal with 3-ms reaction time at 4% CPU load using event-based dynamic vision sensor," *Front. Neurosci.*, vol. 7, p. 223, Nov. 2013.
- [13] A. Glover and C. Bartolozzi, "Event-driven ball detection and gaze fixation in clutter," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2016, pp. 2203–2208.
- [14] A. R. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Ultimate SLAM? Combining events, images, and IMU for robust visual SLAM in HDR and high speed scenarios," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 994–1001, Apr. 2018.
- [15] H. Kim, S. Leutenegger, and A. J. Davison, "Real-time 3D reconstruction and 6-DoF tracking with an event camera," in *Proc. 14th Eur. Conf. Comput. Vis. (ECCV)*, 2016, pp. 349–364. [Online]. Available: http://link.springer.com/10.1007/978-3-319-46466-4_21
- [16] C. Brändli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240 × 180 130 dB 3 μs latency global shutter spatiotemporal vision sensor," *IEEE J. Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, Oct. 2014.
- [17] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, "High speed and high dynamic range video with an event camera," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 6, pp. 1964–1980, Jun. 2021.
- [18] C. Scheerlinck, N. Barnes, and R. Mahony, "Asynchronous spatial image convolutions for event cameras," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 816–822, Apr. 2019.
- [19] H. Patel, C. Iaboni, D. Lobo, J.-w. Choi, and P. Abichandani, "Event camera based real-time detection and tracking of indoor ground robots," 2021, *arXiv:2102.11916*.
- [20] J. Yang et al., "Aircraft tracking based on fully conventional network and Kalman filter," *IET Image Process.*, vol. 13, no. 8, pp. 1259–1265, 2019.
- [21] H. Liu, D. P. Moeyss, G. Das, D. Neil, S.-C. Liu, and T. Delbruck, "Combined frame- and event-based detection and tracking," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCCAS)*, 2016, pp. 2511–2514.
- [22] B. Ramesh et al., "e-TLD: Event-based framework for dynamic object tracking," 2020, *arXiv:2009.00855*.
- [23] S. K. Bose and A. Basu, "A 389TOPS/W, 1262fps at 1meps region proposal integrated circuit for neuromorphic vision sensors in 65nm CMOS," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, 2021, pp. 1–3. [Online]. Available: <https://ieeexplore.ieee.org/document/9634734/>
- [24] S. Paul et al., "A 0.05pJ/pixel 70fps FHD 1meps event-driven visual data processing unit," in *Proc. IEEE Symp. VLSI Circuits*, 2020, pp. 1–2. [Online]. Available: <https://ieeexplore.ieee.org/document/9162948/>
- [25] K. S. Ray, V. K. Asari, and S. Chakraborty, "Object detection by Spatio-temporal analysis and tracking of the detected objects in a video with variable background," 2017, *arXiv:1705.02949*.
- [26] J. Graham, M. Celenk, J. Willis, T. Conley, and H. Eren, "Multiple vehicle tracking using Gabor filter bank predictor," in *Proc. 4th Int. Conf. Comput. Vis. Theory Appl.*, 2009, pp. 632–635.
- [27] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discov. Data Min.*, 1996, pp. 226–231.
- [28] R. Solovveyev, W. Wang, and T. Gabruseva, "Weighted boxes fusion: Ensembling boxes from different object detection models," *Image Vis. Comput.*, vol. 107, Mar. 2021, Art. no. 104117. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0262885621000226>
- [29] "Applications and tools—Metavision SDK docs 4.4.0 documentation," [Online]. Available: <https://docs.prophesee.ai/stable/applications.html>
- [30] M. Fränzl and F. Cichos, "Active particle feedback control with a single-shot detection convolutional neural network," *Sci. Rep.*, vol. 10, no. 1, Jul. 2020, Art. no. 12571. [Online]. Available: <https://www.nature.com/articles/s41598-020-69055-2>
- [31] Q. Cao, Z. Shan, K. Long, and Z. Wang, "GhostCount: A lightweight convolution network based on high-altitude video for vehicle instantaneous counting in dense traffic scenes," *IET Intell. Transp. Syst.*, vol. 17, no. 5, pp. 943–959, 2023. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1049/itr2.12318>
- [32] J. C. Arbeláez, "On object recognition for industrial augmented reality," Ph.D. dissertation, Universidad EAFIT, Medellín, Colombia, 2018, Accepted: Dec. 11, 2019. [Online]. Available: <http://repository.eafit.edu.co/handle/10784/15342>
- [33] T.-R. Tsao and V. Chen, "A neural scheme for optical flow computation based on Gabor filters and generalized gradient method," *Neurocomputing*, vol. 6, no. 3, pp. 305–325, Jun. 1994. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0925231294900671>
- [34] J. Luiten et al., "HOTA: A higher order metric for evaluating multi-object tracking," *Int. J. Comput. Vis.*, vol. 129, no. 2, pp. 548–578, Feb. 2021. [Online]. Available: <https://doi.org/10.1007/s11263-020-01375-2>
- [35] T. Li, W. Romaszkan, S. Pamarti, and P. Gupta, "GEO: Generation and execution optimized stochastic computing accelerator for neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2021, pp. 689–694.
- [36] W. Romaszkan, T. Li, and P. Gupta, "SASCHA—Sparsity-aware stochastic computing hardware architecture for neural network acceleration," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 11, pp. 4169–4180, Nov. 2022. [Online]. Available: <https://ieeexplore.ieee.org/document/9852757/>
- [37] G. Cohen et al., "Event-based sensing for space situational awareness," *J. Astronaut. Sci.*, vol. 66, pp. 125–141, Jun. 2019.
- [38] V. Mohan et al., "EBBINOT: A hardware-efficient hybrid event-frame tracker for stationary dynamic vision sensors," *IEEE Internet Things J.*, vol. 9, no. 21, pp. 20902–20917, Nov. 2022.
- [39] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embedd. Comput. Syst.*, vol. 12, no. 2s, pp. 1–19, 2013.
- [40] P. K. Muthappa, F. Neugebauer, I. Polian, and J. P. Hayes, "Hardware-based fast real-time image classification with stochastic computing," in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, 2020, pp. 340–347.
- [41] V. Schwag, N. Prasad, and I. Chakrabarti, "A parallel stochastic number generator with bit permutation networks," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 65, no. 2, pp. 231–235, Feb. 2018.
- [42] R. K. Budhwani, R. Ragavan, and O. Sentiys, "Taking advantage of correlation in stochastic computing," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2017, pp. 1–4.
- [43] S. Gupta et al., "COSMO: Computing with stochastic numbers in memory," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 2, pp. 1–25, Apr. 2022. [Online]. Available: <https://dl.acm.org/doi/10.1145/3484731>
- [44] J. Yang, A. Graening, W. Romaszkan, V. K. Jacob, P. Gupta, and S. Pamarti, "A 278–514M event/s ADC-less stochastic compute-in-memory convolution accelerator for event camera," in *Proc. IEEE/ACM Int. Conf. VLSI Design*, 2024, pp. 1–2.
- [45] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015, *arXiv:1409.1556*.
- [46] D. Wu, Y. Ruokai, and J. S. Miguel, "Normalized stability: A cross-level design metric for early termination in stochastic computing," in *Proc. 26th Asia South Pac. Design Autom. Conf.*, 2021, pp. 254–259.
- [47] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *Proc. 53rd Annu. Design Autom. Conf. (DAC)*, 2016, pp. 1–6. [Online]. Available: <http://dl.acm.org/citation.cfm?doid=2897937.2898011>
- [48] J.-S. Park et al., "9.5 A 6K-MAC feature-map-sparsity-aware neural processing unit in 5nm flagship mobile SoC," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2021, pp. 152–153.
- [49] C. Deng, Y. Sui, S. Liao, X. Qian, and B. Yuan, "GoSPA: An energy-efficient high-performance globally optimized SParse convolutional neural network accelerator," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2021, pp. 1110–1123.
- [50] J. Lin, Z. Zhu, Y. Wang, and Y. Xie, "Learning the sparsity for ReRAM: Mapping and pruning sparse neural network for ReRAM based accelerator," in *Proc. Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2019, pp. 639–644.
- [51] J. Yue et al., "15.2 A 2.75-to-75.9TOPS/W computing-in-memory NN processor supporting set-associate block-wise zero skipping and ping-pong CIM with simultaneous computation and weight updating," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2021, pp. 238–240. [Online]. Available: <https://ieeexplore.ieee.org/document/9365958/>
- [52] X. Zhang and A. Basu, "A 915–1220 TOPS/W hybrid in-memory computing based image restoration and region proposal integrated circuit for neuromorphic vision sensors in 65nm CMOS," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, 2022, pp. 1–2. [Online]. Available: <https://ieeexplore.ieee.org/document/9772788/>
- [53] C.-X. Xue et al., "16.1 a 22nm 4Mb 8b-precision ReRAM computing-in-memory macro with 11.91 to 195.7TOPS/W for tiny AI edge devices," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2021, pp. 245–247. [Online]. Available: <https://ieeexplore.ieee.org/document/9365769/>
- [54] P. Deaville, B. Zhang, and N. Verma, "A 22nm 128-kb MRAM row/column-parallel in-memory computing macro with memory-resistance boosting and multi-column ADC readout," in *Proc. IEEE Symp. VLSI Technol. Circuits (VLSI Technol. Circuits)*, 2022, pp. 268–269. [Online]. Available: <https://ieeexplore.ieee.org/document/9830153/>
- [55] S. Jung et al., "A crossbar array of magnetoresistive memory devices for in-memory computing," *Nature*, vol. 601, no. 7892, pp. 211–216, Jan. 2022. [Online]. Available: <https://www.nature.com/articles/s41586-021-04196-6>