GEO: Generation and Execution Optimized Stochastic Computing Accelerator for Neural Networks

Tianmu Li, Wojciech Romaszkan, Sudhakar Pamarti, Puneet Gupta

Electrical and Computer Engineering Department
University of California, Los Angeles
Los Angeles, USA
{litianmu1995, wromaszkan, spamarti, puneetg}@ucla.edu

Abstract—**Stochastic computing (SC) has seen a renaissance in recent years as a means for machine learning acceleration due to its compact arithmetic and approximation properties. Still, SC accuracy remains an issue, with prior works either not fully utilizing the computational density or suffering from significant accuracy losses. In this work, we propose GEO – Generation and Execution Optimized Stochastic Computing Accelerator for Neural Networks, which optimizes stream generation and execution components of SC, and bridges the accuracy gap between stochastic computing and fixed-point neural networks. It improves accuracy by coupling controlled stream sharing with training and balancing OR and binary accumulations. GEO further optimizes the SC execution through progressive shadow buffering and architectural optimizations. GEO can improve accuracy compared to state-of-the-art SC by 2.2-4.0% points while being up to 4.4X faster and 5.3X more energy efficient. GEO eliminates the accuracy gap between SC and fixed-point architectures while delivering up to 5.6X higher throughput and 2.6X lower energy.**

I. Introduction

The rapid growth of deep learning in the past decade has created an immense demand for computing power at both the cloud and edge. Multiple algorithmic, architectural, and circuit approaches have been proposed to meet this demand. Among those, stochastic computing (SC) has been enjoying a renaissance in deep learning acceleration for latency-, energy-, and cost-constrained devices [1]–[5]. It offers a very compact computing footprint, enabling high levels of parallelism and data reuse not achievable using conventional floating- or fixed-point architectures [5]. Its approximate nature synergizes well with neural networks' inherent error-tolerant properties, enabling new axes of accuracy and performance tradeoffs [3], [5], [6].

Stochastic computing represents numbers using the proportion of ones in a random bitstream and enables multiplication and addition using simple logic gates. Precision remains the most significant barrier towards wider SC adoption; therefore, the majority of prior works opted for approximate parallel counter-based accumulation fabric [4], [6]–[8] or directly converting each multiplication result and adding them in the fixed-point domain [3], [9], losing computational density. Custom SC addition circuits have also been used [10], [11]. [5] showed OR-accumulation using split-unipolar stochastic streams to be a viable, unscaled accumulation approach for neural network acceleration, but it suffers from significant accuracy loss. In this work, we show that those sacrifices are not necessary.

Stochastic bitstream generation has also received much attention. A typical stochastic number generator (SNG) has a random number generator (RNG) and a comparator that compares the target value with the random number [12]. Streams generated from a true random number generator (TRNG) have a predictable error variance that can only be reduced through longer stream lengths [13]. Less expensive TRNGs [14]–[17] as well as quasi-random sequences [3], [4], [9] have been explored to reduce error and cost of stream generation. Correlation of the random sources in stream generation can severely impact accuracy and has forced most prior works to limit the amount of computation performed in the stochastic domain, sacrificing potential performance benefits [3], [4], [9].

Most SC literature focuses on SC "component" improvements [3], [9], [18] or implement dedicated network-specific accelerators [7], [19]. Programmable, full-system SC implementations [2], [5], like the focus of our work are rare. We account for overheads of generalizability of programmable accelerators and generate power, performance, and accuracy numbers for the entire compute+memory system.

We propose GEO - Generation and Execution Optimized stochastic computing for neural networks - an ensemble of optimization techniques that can bridge the accuracy gap between stochastic and fixed-point accelerators while improving inference energy and latency even when compared to state of the art stochastic inference accelerators. Our contributions are as follows:

- We show that, with appropriate training, neural networks can learn the biases caused by the use of pseudo-RNGs and extensive sharing of them in SNGs and *improve* accuracy compared to using non-shared TRNGs by as much as 6.1% points while reducing energy and area.
- We propose a progressive stream generation and shadow buffering scheme that reduces required memory bandwidth by up to 4X while improving latency by as much as 2X.
- We propose using a balanced mix of stochastic OR and fixed-point accumulation to improve accuracy by up to 9.4% points. The increase in accuracy allows us to reduce stream length by 4X while still maintaining 2.2-4.0% points accuracy advantage.
- We leverage aggressive pipelining and near memory computation to enable high throughput, maximal reuse, and efficient compute utilization regardless of layer parameters.

II. Stochastic Stream Generation Optimizations

This section proposes two methods optimizing the stream generation process of SC. Combining shared stream generation and training improves accuracy, while progressive generation relieves memory bottleneck.

A. Co-optimized Shared Generation and Training

RNG Sharing has been shown to be detrimental to stochastic computing accuracy [20], [21], and typically requires complicated methods to decorrelate streams from the same source to avoid incurring large stream generation penalties. However, we hypothesize that a partially-shared generation leads to higher accuracy, especially when coupled with deterministic stream generation and stream-based training.

Deterministic and repeatable (using a pseudorandom RNG) stream generators guarantees obtaining the same outputs from the same inputs, enabling the model to train for a fixed, instead of random error. We achieve determinism using maximal-length linear feedback shift registers (LFSR) as RNG. When generating streams of length 2^n, an n-bit maximal-length LFSR is used with a cycle of $2^n - 1$. Apart from guaranteeing an almost accurate generation, LFSR generates the same output with the same input and seed and allows multiple uncorrelated stream generation (by varying the seed or the characteristic polynomial) suitable for large multiply-accumulate operations. Sharing stream generation simplifies the error profile caused by SC. Assuming that all kernels in a layer share the same set of seeds, training only needs to deal with an error associated with one set of seeds.

To test this hypothesis, we implement three levels of sharing for a 4-layer CNN [22] on the SVHN dataset. Streams are represented using the split-unipolar format, and OR is used for accumulation, similar to [5]. In the "no sharing" case, each SNG gets a different seed for its LFSR. The "moderate sharing" case shares the same set of seeds across all kernels in a given layer. Finally, in the "extreme sharing" case, all rows of all kernels in a layer use the same set of seeds. The same is done when a true random number generator (TRNG) is used as an RNG[1]. The results are shown in Figure 1. *At moderate sharing levels, LFSR-based SNGs show a significant uplift in the accuracy (up to 6.1% points compared to unshared TRNGs) at both stream lengths*, adhering to the hypothesis. TRNG does not see the accuracy improvement with sharing due to the lack of determinism. However, both TRNG and LFSR suffer from a significant drop in accuracy when using extreme sharing. In this case, stream correlation becomes an issue hard to overcome just by training.

These results also mean that low discrepancy (LD) sequences are not suitable for OR accumulation due to the difficulty of generating multiple uncorrelated streams, even though LD sequences can improve accuracy for single operations [23]. We also compared the validation accuracy when using LFSR without modeling it during training. The models are trained using TRNG, but validated using LFSR. No accuracy can be gained from moderate sharing when the model is not trained for it, and extreme sharing reduces accuracy to about 20%. We use the moderate sharing scheme in GEO (up to the limit of availability of unique RNG seeds).

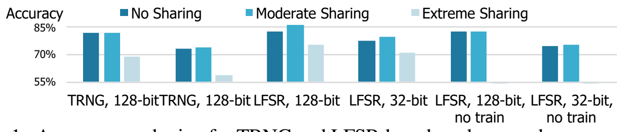

Fig. 1. Accuracy vs. sharing for TRNG and LFSR-based random number generation.

B. Progressive Stochastic Stream Generation

Once a set of weights and activations finish computation, the SNG buffers need to be reloaded for the next iteration of generation and computation. If the underlying architecture needs to reload activations and weights extensively during computation, reloading can become a significant bottleneck. We propose using a progressive generation scheme to alleviate this inefficiency, where stream generation begins as soon as the first 2 most-significant bits are loaded into the buffers instead of waiting for all 8 bits, as shown in Figure 3. The rest of the buffer is padded with 0s. As stream generation continues, the remaining bits are gradually loaded in groups of 2 bits every two cycles until the number of bits loaded matches the LFSR length used. As GEO matches the LFSR length to the stream length being used, shorter stream lengths effectively truncate the converted fixed-point values. Our progressive buffering scheme can take advantage of that truncation to reduce the number of required memory accesses, which is not possible when all bits for a given value are being loaded in parallel. Compared to starting generation when all 8-bits are loaded, *progressive generation reduces the latency overhead of reloading by 4X.*

[1]Due to the lack of hardware TRNG, we approximate it using the rand function in PyTorch

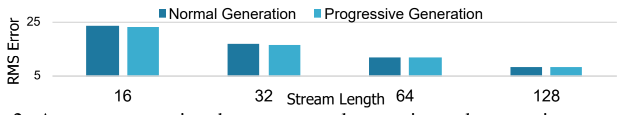

Fig. 2. Accuracy comparison between normal generation and progressive generation performing a multiplication of two uniformly sampled inputs. RMS Error is multiplication error compared to an 8-bit integer.

As shown in Figure 2, performing progressive loading does not hurt multiplication accuracy. Generation is accurate after eight cycles at most when the loaded values match LFSR length. Progressive loading introduces error in at most 8 cycles when using 7-bit lfsr and 128-bit streams. On a network level using the same setup as Section II-A on SVHN, using progressive loading only lowers accuracy by 0.42% when using 32-bit streams and 0.16% when using 64-bit streams. Note that this is a worst-case scenario where all input and weight streams are assumed to be generated progressively. Any input or weight reuse in the architecture leads to fewer reloads.

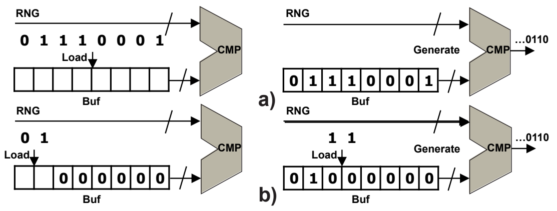

Fig. 3. Normal SNG (a) and progressive stream generation (b).

III. Stochastic Computing Execution Optimizations

This section describes the overall GEO architecture and discusses a variety of micro-architectural optimizations to improve performance and accuracy on GEO.

A. GEO Architecture

Before describing further execution optimizations, we will briefly explain the underlying accelerator architecture. The GEO accelerator uses fully-stochastic computation, which can easily be modified to support different levels of partial-binary accumulation. Further, it is agnostic to the stream generation scheme and supports extensive RNG sharing. We will now briefly describe the architecture functionality.

Figure 4 a) shows the block diagram of the accelerator. It uses separate *weight and activation memories*, which are used to load their respective *SNG buffers*. Both weight and activation memories are organized in 2 logical banks, supporting ping-pong operation. For weights, this allows loading the next set of kernels from external memory, while the current one is being processed. For activations, it enables loading activations while writing back partial sums and outputs. Both sets of memories are sized accordingly to support such operation.

Once all required inputs and weights are loaded into the buffers, the stream generation begins, and SNG outputs are fed directly into the compute engine. The compute is organized to maximize density while minimizing the conversion costs of stochastic streams. It is logically partitioned into *rows*, where each row is responsible for one output channel. This way, the same set of activations can be broadcasted across multiple rows, amortizing activation stream generation costs. Due to the benefits of sharing seeds between kernels shown in Sec. II-A, different rows share the same set of LFSR. Within each row, the same set of weights is multiplied with different sets of activations, emulating the convolutional sliding window. This way, the architecture also achieves high levels of weight reuse.

Output streams of each row are passed to the *output converter array*, where individual *output converter* modules convert them to a fixed-point format to accumulate the final value into a counter. By using small, configurable parallel counters before the conversion, the output

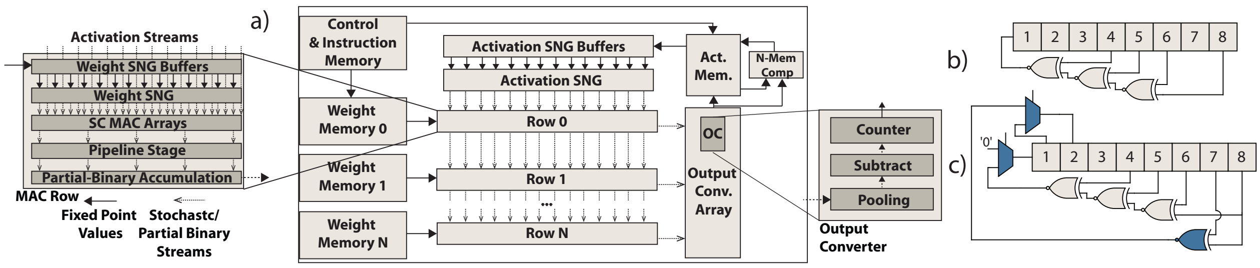

Fig. 4. Overall SC accelerator architecture block diagram. with breakdowns of the MAC row (left) and output converter (right) modules (a). Fixed 8-bit maximum length LFSR (b), and configurable 8- or 7-bit maximum length LFSR (c).

converter array can add together neighboring outputs, achieving average pooling with computation skipping on layers followed by pooling operators as in [5]. Computation skipping allows shorter streams on layers with pooling since average pooling adds together multiple streams in fixed-point. Once the stream generation is finished and output values are completely accumulated, they are passed through the near-memory batch normalization and ReLU activation blocks, before being written back to activation memory to serve as inputs to the next layer.

Despite its rigid compute engine, the architecture can support a wide variety of activation and kernel sizes, as well as padding and pooling for convolutional layers. It supports fully-connected (FC) layers, although with compute underutilization. It is also fully programmable, with its own ISA and instruction memory.

B. Partial Binary Accumulation

As mentioned in Section I, many recent SC works opt to perform accumulation in the fixed-point domain, as it offers higher accuracy than SC-based addition [9], [19]. In contrast, a few others have tried implementing fully-stochastic accumulation to save costs. In contrast to these two extremes, we propose to use partial SC-fixed-point accumulation, where the first few levels of accumulation are implemented in SC using OR gates, before converting the intermediate results to fixed-point and computing the remainder of the accumulation.

The partition between SC and fixed-point accumulation significantly affects both accuracy and performance. While using an approximate parallel counter (APC) [24] allows one layer of SC accumulation before fixed-point accumulation, the combined use of AND and OR makes it equivalent to multiplexers and is thus unsuitable for multiple layers of accumulation. Using OR for accumulation with training allows an arbitrary trade-off between SC and fixed-point accumulation. We tested model accuracy with different fixed-point accumulation levels using the same setup as in Sec. II-A. Assuming weight filters are arranged into (C_{in}, H, W) dimensions, *performing fixed-point accumulation in the W dimension (PBW) improves accuracy by 4.5% and 9.4% respectively for 128-bit and 32-bit streams compared to performing all accumulations using OR*. Extending fixed-point accumulation to H (PBHW) as well improves accuracy by $< 0.5\%$ but increases the number of fixed-point adders by 5X for 5×5 filters.

Adding support for partial binary accumulation only requires replacing the last levels of OR-accumulation with a parallel counter. While the level of partial binary accumulation is fixed at the design stage, it still allows for trading off precision with latency through SC stream length configuration. Since partial binary accumulation fabric operates on a bitwise basis, it is agnostic to the chosen stream length. Parallel counters in the average pooling fabric in the output converters need to be adjusted to handle wider inputs. In Section IV, we show that those changes have minimal impact on the overall architecture.

Figure 5 shows the overhead, in terms of area, of implementing SC-based MAC units with partial binary accumulation stages. We compare the full-or accumulation (SC), PBW, PBHW, and fixed-point accumulation (FXP) configurations, for different three-dimensional kernel sizes. While area overhead of PBW and PBHW partial binary accumulation can be as much as 1.4X and 4.5X for smaller kernels, the area increase goes down to 4% and 9% for large ones. Implementing partial binary accumulation is therefore well suited for highly-parallel SC architectures where such overheads would be negligible. Figure 5 also shows that implementing complete binary accumulation can increase the area by more than five times for most kernel sizes, emphasizing its performance limitation. While approximate parallel counters [24] (APC) offers noticeable area benefits compared to fixed-point accumulators, it is still more than 3X larger than PBW and PBHW for larger kernels. Given that PBW is almost identical accuracy-wise, the rest of the paper uses PBW as the default unless otherwise mentioned.

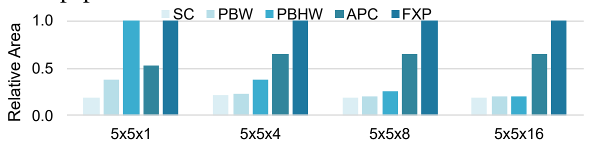

Fig. 5. Area comparison for different hardware implementations of SC-based MAC units for different kernel sizes and different levels of partial binary accumulation.

Using partial binary accumulation increases the dynamic range of outputs. Since the increase of output precision comes primarily from increased range, truncating activations without factoring in the dynamic range diminishes partial binary accumulation benefits. To deal with this, we use an 8-bit fixed-point version of batch normalization (BN) before ReLU activation to minimize the cost of implementing it in hardware. While still potentially expensive, BN offers 5.5-6.5% points accuracy improvement. For layers with pooling, pooling is placed before ReLU activations, so that BN can be performed on pooled activations.

C. Near-Memory Computation

Organizing the GEO accelerator compute hierarchy to mimic a vertically sliding convolutional window means that it naturally yields to the weight-stationary dataflow [25]. While the window iterates through the output tensor, weights can stay unchanged, and only a single row of activations needs to be reloaded between each computational pass, therefore minimizing both weight and activation bandwidth requirements. Indeed, this dataflow choice reduces the overall number of memory accesses by up to 3.3X compared to input-stationary, making it the optimal choice in virtually every convolutional layer we have explored. However, this is only true if a strict, weight-stationary implementation can be enforced. It requires that the MAC units' width and the corresponding number of SNGs are sized to fit the entire activation tensor covered by a kernel in a given layer. This constraint guarantees that output values can be generated during a single computational pass, without partial sums, effectively "merging" weight- and output-stationary dataflows in one.

However, it is not uncommon in modern neural networks to find kernels with thousands of weights, which cannot be fully unrolled without sacrificing a prohibitive amount of silicon area. When that is not possible, the accelerator needs to store converted partial sums for later accumulation. If the architecture does not support that, it has to implement a strict output-stationary dataflow, accumulating output values in output conversion modules over multiple passes, where both weights and activations need to be swapped between each pass. Such dataflow can increase memory accesses by as much as 10.3X vs. ideal, weight-stationary implementation. While progressive generation can alleviate such dataflow's bandwidth requirements to a degree, the steep energy cost of memory accesses remains.

One way to avoid being forced into such suboptimal dataflow is to couple output conversion modules with small register files. However, the number of registers required will depend on a particular layer - to support some of the very deep ones would require register files that dwarf the size of conversion modules. At the same time, those register files would remain mostly unused on the shallower layers. Instead, *we propose implementing near-memory accumulation, where the activation memory is tightly coupled with an array of adders*. We then expand the GEO ISA to support a 2-cycle read-add-write vector instruction that can be used to accumulate partial sums. Since partial sums are stored in large activation memory, there is no need to size it for any specific network or layer.

There are two downsides w.r.t. to local register files. First, activation memory accesses are much more energy costly than to local registers. However, in this dataflow, partial sum accesses constitute only 13% to 20% of overall memory accesses, meaning they are not critical to overall energy consumption. Second, additional accesses put more strain on memory bandwidth. However, as we will show in Section IV, progressive shadow buffering can alleviate this problem. We further expand this scheme to support near-memory batch normalization through an array of fixed-point MAC units, tightly coupled with activation memory.

D. Pipeline Optimizations

On top of the generation optimizations from Section II and execution optimizations listed above, the GEO accelerator includes two microarchitectural enhancements. First, we supplement the progressive generation with shadow buffers. When current progressive values are fully loaded, a certain number of bits can be loaded into the shadow buffers for the next computation. Thanks to that, the following computation phase can begin immediately after the current one finishes, since the minimum number of bits required, which in our case is 2, is already available in the shadow buffer. Without progressive generation, shadow buffers would need to be the same size as the actual SNG buffers (i.e., 4X larger), incurring significant area penalty. The overhead of progressive shadow buffers is only about 4% at the whole accelerator level.

Second, we implement a pipeline stage within our compute engine between the SC and partial-binary accumulation stages. This is because of a long critical path between the LFSR, SNG, SC MAC, partial binary accumulation, and output counters. *Implementing the pipeline stage in that location allows us to cut down the critical path by over 30% while minimizing the area required by additional flip-flops* ($<1\%$ overhead on the accelerator level). Because of the recovered timing slack, we can now reduce the operating voltage without lowering the frequency to achieve better energy efficiency.

IV. Evaluation & Results

We test accuracies on CIFAR-10, SVHN, and MNIST datasets. For CIFAR-10 and SVHN, we use the same 4-layer CNN [22] (CNN-4) as in Section III and VGG-16 [26]. VGG-16 has the X/Y input dimensions of each layer downscaled, and the fully-connected layers reduced to FC-512 instead of FC-4096 to accommodate the smaller image sizes. For MNIST we use LeNet-5 [27]. We use PyTorch 1.5.0 to train the models. We implement the forward pass using both floating-point and simulated SC. Simulated SC is used to compute output values, while the floating-point forward pass is used to guide back propagation. With SC simulation's speed limitations, we skipped training for more complex datasets (i.e., ImageNet) due to the prohibitively long training time. Due to the use of floating point for back propagation, GEO can only accelerate inference of SC models. Models are trained using ADAM optimizer with an initial learning rate of 2e-3, and accuracy is evaluated on the corresponding testing dataset after 1000 epochs. Each model is trained with different stream lengths using split-unipolar implementation, and designated by two stream lengths $\{s_p - s\}$, s_p for layers with pooling and s for layers without. While max pooling is possible, we use average pooling with computation skipping to reduce stream length requirements for layers with pooling. Output layers always use 128-bit streams due to their small performance impact but noticeable accuracy benefits. The actual stream length used is double the specified value due to the use of split-unipolar representation.

To estimate the area, power, and latency of the proposed design, we have written individual blocks (SNGs, MAC arrays, buffers, output converters) in Verilog, and then synthesized them using a commercial 28nm HVT library. Memories were modeled using CACTI 6.5 [28]. For the LP variant desribed below, we consider the cost of external memory accesses, with the bandwidth and access energy modeled after the HBM2 standard [29]. We used activity factors obtained through RTL simulations to adjust active power numbers in synthesis (since many modules, such as SNG buffers and batch normalization modules are idle most of the time). To obtain accurate energy and latency estimates, we used a custom performance simulator, which combines the numbers from individual modules with a compiled code representing the given network model. Since the proposed enhancements are mostly agnostic of the control flow, we use the ISA proposed in [5] with minor modifications. We create two versions of GEO: ultra-low power (ULP) or low-power (LP) targeted at different area-points and network sizes. ULP has 25.6K MACs with total on-chip memory of 150KB, while LP variant has 294K MACs and 0.5MB of on-chip memory.

As a fixed-point baseline, we use Eyeriss [25], scaled to 4-bit or 8-bit precision and 28nm node. The on-chip memory capacity and the number of processing elements are chosen to achieve close to iso-area comparison point with GEO. We simulate the execution of the neural networks using [30]. For SC comparison points, we use the ACOUSTIC [5], Sign-Magnitude SC (SM-SC) [1] and SCOPE [2]. ACOUSTIC configurations are sized to have the same amount of memory and compute as GEO, and we use longer stream lengths to maintain close to iso-accuracy with GEO. ACOUSTIC architecture configurations differ from the original, but we use the same simulation framework, ensuring consistent results. SM-SC is not a fully programmable accelerator making full comparison impossible. SCOPE is an in-memory, DRAM-based accelerator with a massive area footprint, not well suited towards edge applications [2]. Unfortunately, many recent works on SC neural network acceleration only report performance numbers for the compute part, while omitting the crucial impact of memory and dataflow, making it impossible for us to compare on the system level. Numbers are scaled to the 28nm node when necessary, using the models provided in [31]. We further compare GEO-ULP with CONV-RAM [32] and MDL-CNN [33] mixed-signal accelerators.

TABLE I
ACCURACY COMPARISON WITH FIXED-POINT, OTHER SC IMPLEMENTATIONS AND SO ON.

Dataset	Model	Eyeriss 8-bit	Eyeriss 4-bit	ACOUSTIC [5] 256	ACOUSTIC [5] 128	GEO 64-128	GEO 32-64	GEO 16-32	SCOPE [2] 128	CONV-RAM [32] 7a1w	MDL-CNN [33] 4a1w	SM-SC [1] 128
CIFAR-10	CNN-4	85.1%	82.1%	78.0%	74.9%	**80.2%**	**78.1%**	—	—	—	—	80%
	VGG-16	90.9%	—	—	—	**88.7%**	**88.7%**	—	—	—	—	—
SVHN	CNN-4	93.3%	90.5%	89.0%	86.8%	**91.9%**	**90.8%**	—	—	—	—	—
	VGG-16	96.2%	—	—	—	**96.0%**	**95.9%**	—	—	—	—	—
MNIST	LeNet-5	—	99.3%	—	99.3%	—	**99.3%**	**98.9%**	99.3%	96%	98.4%	—

To ease future comparisons and benchmarking, we will open-source our SC training code (heavily optimized for stream-based training on GPUs and CPUs) and the GEO architecture simulator at https://github.com/nanocad-lab/geo.

A. GEO Accuracy Comparisons

Table I compares accuracy of GEO with fixed-point and other SC implementations. Eyeriss results are retrained at respective precision. [2] Results for other works are reported from the respective papers. *GEO offers 2.2-4.0% points better accuracy at quarter stream length compared to [5] and similar accuracy at the same stream length compared to [1].* Both shared stream generation and partial binary accumulation contribute to increased accuracy. For CNN-4 on SVHN with 32-64 stream length, dropping binary accumulation lowers accuracy to 79.6%, while using TRNG on top of that drops it further to 73.7%. Compared to fixed-point, the accuracy with CNN-4 is comparable to 4-bit fixed-point when using 32-64 setup on SVHN, but 4% lower on CIFAR-10 when using 32-64 and 1.9% lower when using 64-128[3]. Accuracy with VGG-16 is 2.2% lower than 8-bit fixed-point on CIFAR-10 and comparable on SVHN. Accuracy on MNIST is already comparable to fixed-point in the baseline, and GEO optimizations don't affect it. Compared to CONV-RAM [32], an in-memory architecture and MDL-CNN [33], a time-domain architecture, GEO offers superior accuracy even with 16-32 stream length.

B. Performance Impact of GEO Enhancements

We compare the baseline ULP architecture (without GEO optimizations and 16-bit LFSRs to emulate TRNG) with two GEO variants::

- GEO-GEN-128,128 - uses the generation optimizations from Section II. Progressive shadow buffers are used in this configuration.
- GEO-GEN-EXEC-32,64 - uses both the generation and execution (from Section III optimizations. Further, it reduces the stream lengths being used to remain iso-accuracy with other configurations.

Area, energy, and latency impacts of GEO optimizations on the ULP architecture are shown in Figure 6. For energy and latency, we simulated the SVHN CNN inference on each of those design points. Generation optimizations result in an overall 1% decrease in the accelerator area, where an increase in area due to progressive shadow buffers is balanced by more extensive RNG sharing. At the same time, *the use of progressive shadow buffers to hide memory latency results in a 1.7X speedup and 1.6X reduction in energy*. Energy savings come mainly from SNG optimizations and reduced leakage.

Adding execution optimizations on top of the generation ones increases the area by 2% w.r.t. to baseline. The impact of pipelining and partial binary accumulation is minimal due to its limited application and an overall small contribution of the SC MAC array to the overall area. Similarly, near-memory computation is well amortized because it is time multiplexed. *The combination of shorter stream lengths, more efficient dataflow enabled by near-memory computation and pipelining coupled with DVFS results in 4.3X and 5.2X reduction in latency and energy w.r.t. baseline.*

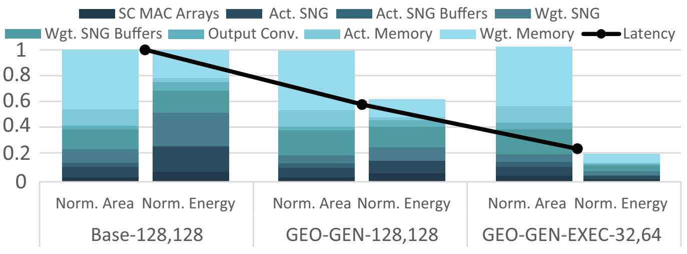

Fig. 6. Area, energy and latency for different GEO configurations (normalized to Base-128,128).

C. GEO Performance Compared

Table II shows a comparison of the proposed GEO ULP accelerator with fixed-point and mixed-signal approaches. First, we show that GEO-32,64 outperforms the 4-bit fixed-point baseline in terms of throughput, by 2.7X, and energy efficiency, by 2.6X, in the same area. It also outperforms ACOUSTIC-128, by 4.4X and 5.3X, respectively, while achieving higher accuracy. It is also highly-competitive in terms of energy-efficiency with mixed-signal accelerators like Conv-RAM and MDL-CNN. We refrain from comparing the throughput against those implementations due to the large area difference.

TABLE II
COMPARISON BETWEEN GEO ULP AND FIXED-POINT AND NEUROMORPHIC IMPLEMENTATIONS. NUMBERS ARE SCALED TO 28NM.

	Eyeriss 4-bit [25]	**GEO ULP -32,64**	Conv-RAM [32]	MDL CNN [33]	ACOUSTIC ULP-128 [5]	**GEO ULP -16,32**
Voltage	0.9	**0.81**	0.9	0.537	0.9	**0.81**
Area [mm^2]	0.59	**0.58**	0.02	0.06	0.57	**0.58**
Power [mW]	20	**48**	0.016	0.02	72	**48**
Clock [MHz]	400	**400**	364	25	400	**400**
Precision	4-bit	—	6b/1b	8b/1b	—	—
CIFAR-10 Fr/s	5.2k	**14k**	—	—	3.2k	**29k**
CIFAR-10 Fr/J	115k	**305k**	—	—	57k	**576k**
LeNet5 CNN Fr/s	47k	**520k**	15k	1k	3.2k	**780k**
LeNet5 CNN Fr/J	790k	**42M**	117M	50M	57k	**56M**
Peak GOPS[4]	80	**640**	10.7	0.365	160	**1280**
Peak TOPS/W	4	**13.3**	44.2	18.2	2.22	**26.6**

On the scale-out end of the spectrum, GEO LP outperforms iso-area, 8-bit Eyeriss by 5.6X in terms of throughput and 2.6X in terms of energy efficiency. Modest energy reduction is caused by the high cost of external memory accesses - when those are omitted, GEO is as much as 6.1X more energy-efficient than Eyeriss. It is also 2.4X faster and 1.6X more energy efficient than ACOUSTIC, while having higher inference accuracy. Despite occupying only 3.3% of SCOPE area, GEO has nearly 24% of its peak throughput and has 2.4X better energy efficiency.

V. Conclusion

In this paper, we present GEO, a generation, and computation-optimized stochastic computing architecture for neural network

[2]Original Eyeriss [25] was 16-bit with truncated accumulation which suffers from substantial accuracy loss at lower 4/8-bit precision. We assume full 16 bit accumulation bitwidth and as a result Eyeriss accuracy results are somewhat optimistic.

[3]While intermediate accumulation results for Eyeriss are allowed to have double the input precision, overflow may still happen and these results are optimistic

TABLE III
COMPARISON BETWEEN GEO LP AND FIXED-POINT AND SC IMPLEMENTATIONS. NUMBERS ARE SCALED TO 28NM.

	Eyeriss 8-bit [25]	**GEO LP -64,128**	SM-SC [1]	SCOPE [2]	ACOUSTIC LP-256 [5]	**GEO LP -32,64**
Voltage	0.9	**0.81**	0.9	——	0.9	**0.81**
Area [mm^2]	9.3	**9.2**	——	273	9	**9.2**
Power [mW]	848	**797**	——	——	1160	**797**
Clock [MHz]	400	**400**	1536	200	400	**400**
CIFAR-10 VGG Fr/s	555	**3.1k**	——	——	1.3k	**5.2k**
CIFAR-10 VGG Fr/J	618	**1.6k**	——	——	1k	**2.2k**
Peak GOPS	204	**1.8k**	1.7k	7.1k	460	**3.6k**
Peak TOPS/W	0.48	**2.25**	0.92	——	0.4	**4.5**

acceleration. We develop an ensemble of accuracy improvement (co-optimized stream generation and training, partial binary accumulation) and energy/runtime improvement (progressive stream generation, near memory computation, shadow buffering and pipelining) techniques. These optimizations improve accuracy by 2.2-4.0% points compared to state of the art SC-based accelerators while also being 4.4X faster and 5.6X more energy efficient. GEO can compete with fixed-point implementations with similar accuracy and area while delivering up to 5.6X throughput, 2.6X energy-efficiency gains. GEO, despite being an all-digital, programmable accelerator can achieve energy efficiency comparable to in-memory/mixed-signal accelerators. Our ongoing work focuses on taping out a silicon prototype of GEO-ULP and developing fast training approaches for stochastic computing.

Acknowledgment

This material is based on research sponsored by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-27867. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defense Advance Research Projects Agency (DARPA) or the U.S. Government.

References

[1] A. Zhakatayev, S. Lee, H. Sim, and J. Lee, "Sign-Magnitude SC : Getting 10X Accuracy for Free in Stochastic Computing for Deep Neural Networks ," *DAC*, pp. 1–6, 2018.

[2] Y. Li, Shuangchen and Oliver Glova, Alvin and Hu, Xing and Gu, Peng and Niu, Dimin and T. Malladi, Krishna and Zheng, Hongzhong and Brennan, Bob and Xie, "SCOPE: A Stochastic Computing Engine for DRAM-based In-situ Accelerator," in *IEEE Micro*, 2018, pp. 697–710.

[3] R. Hojabr, K. Givaki, S. M. R. Tayaranian, P. Esfahanian, A. Khonsari, D. Rahmati, and M. H. Najafi, "SkippyNN : An Embedded Stochastic-Computing Accelerator for Convolutional Neural Networks," in *DAC*, 2019, pp. 1–6.

[4] S. R. Faraji, M. H. Najafi, B. Li, D. J. Lilja, and K. Bazargan, "Energy-Efficient Convolutional Neural Networks with Deterministic Bit-Stream Processing," in *DATE*, 2019, pp. 1757–1762.

[5] W. Romaszkan, T. Li, T. Melton, S. Pamarti, and P. Gupta, "ACOUSTIC : Accelerating Convolutional Neural Networks through Or-Unipolar Skipped Stochastic Computing," in *DATE*, 2020.

[6] Z. Yawen, Z. Xinyue, S. Jiahao, W. Yuan, H. Ru, and W. Runsheng, "Parallel Convolutional Neural Network (CNN) Accelerators Based on Stochastic Computing," in *SiPS*, 2019, pp. 19–24.

[7] Z. Li, J. Li, A. Ren, R. Cai, C. Ding, X. Qian, J. Draper, B. Yuan, J. Tang, Q. Qiu, and Others, "HEIF: Highly Efficient Stochastic Computing based Inference Framework for Deep Neural Networks," *IEEE TCAD*, vol. 0070, no. c, pp. 1–14, 2018.

[8] B. Li, M. H. Najafi, and D. J. Lilja, "Low-Cost Stochastic Hybrid Multiplier for Quantized Neural Networks," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 2, 2019.

[9] H. Sim and J. Lee, "A New Stochastic Computing Multiplier with Application to Deep Convolutional Neural Networks," pp. 1–6, 2017.

[10] D.-a. Nguyen, H.-h. Ho, D.-h. Bui, and X.-t. Tran, "An Efficient Hardware Implementation of Artificial Neural Network based on Stochastic Computing," in *NICS*. IEEE, 2018, pp. 237–242.

[11] B. Li, S. Koester, and D. J. Lilja, "Low Cost Hybrid Spin-CMOS Compressor for Stochastic Neural Networks," in *GLSVLSI*, 2019, pp. 141–146.

[12] B. R. Gaines, "Stochastic computing systems," in *Advances in information systems science*, 1969, pp. 37—-172.

[13] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *DATE*. Leuven, BEL: European Design and Automation Association, 2014.

[14] X. Ma, L. Chang, S. Li, L. Deng, Y. Ding, and Y. Xie, "In-memory multiplication engine with SOT-MRAM based stochastic computing," *CoRR*, vol. abs/1809.0, 2018.

[15] A. Mondal and A. Srivastava, "Data driven optimizations for MTJ based stochastic computing," *CoRR*, vol. abs/1804.03228, 2018.

[16] M. W. Daniels, A. Madhavan, P. Talatchian, A. Mizrahi, and M. D. Stiles, "Energy-Efficient Stochastic Computing with Superparamagnetic Tunnel Junctions," *Physical Review Applied*, no. 3, p. 1.

[17] S. Wang, S. Pal, T. Li, A. Pan, C. Grezes, P. Khalili-Amiri, K. L. Wang, and P. Gupta, "Hybrid vc-mtj/cmos non-volatile stochastic logic for efficient computing," in *DATE*, 2017, pp. 1438–1443.

[18] D. Wu, J. Li, R. Yin, H. Hsiao, Y. Kim, and J. S. Miguel, "Ugemm: Unary computing architecture for gemm applications," in *ISCA*, 2020, pp. 377–390.

[19] A. Ren, J. Li, Z. Li, C. Ding, X. Qian, Q. Qiu, B. Yuan, and Y. Wang, "SC-DCNN: Highly-Scalable Deep Convolutional Neural Network using Stochastic Computing," *CoRR*.

[20] H. Ichihara, S. Ishii, D. Sunamori, T. Iwagaki, and T. Inoue, "Compact and accurate stochastic circuits with shared random number sources," *ICCD*, pp. 361–366, 2014.

[21] F. Neugebauer, I. Polian, and J. P. Hayes, "Building a better random number generator for stochastic computing," in *DSD*, 2017, pp. 1–8.

[22] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: efficient neural network kernels for arm cortex-m cpus," *CoRR*, vol. abs/1801.06601, 2018.

[23] S. Liu and J. Han, "Energy Efficient Stochastic Computing with Sobol Sequences," in *DATE*. EDAA, 2017, pp. 650–653.

[24] K. Kim, J. Lee, and K. Choi, "Approximate de-randomizer for stochastic circuits," in *ISOCC*, 2015, pp. 123–124.

[25] Y. H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *Proc. ISSCC*, vol. 52, no. 1, pp. 262–263, 2016.

[26] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, pp. 1–14.

[27] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[28] H. Labs, "CACTI-6.5 (Cache Access Cycle Time Indicator)." [Online]. Available: http://www.hpl.hp.com/research/cacti/

[29] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-grained dram: energy-efficient dram for extreme bandwidth systems," in *MICRO*. IEEE, 2017, pp. 41–54.

[30] M. Gao and M. Horowitz, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," in *ASPLOS*, 2017, pp. 751–764.

[31] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm," *Integration, the VLSI Journal*, no. January, pp. 74–81.

[32] A. Biswas and A. P. Chandrakasan, "Conv-ram: An energy-efficient sram with embedded convolution computation for low-power cnn-based machine learning applications," in *ISSCC*, 2018, pp. 488–490.

[33] A. Sayal, S. Fathima, S. S. T. Nibhanupudi, and J. P. Kulkarni, "All-Digital Time-Domain CNN Engine Using Bidirectional Memory Delay Lines for Energy-Efficient Edge Computing," *ISSCC*, vol. 49, no. 4, pp. 228–230, 2019.