A Near-Sensor Processing Accelerator for Approximate Local Binary Pattern Networks

Shaahin Angizi, *Senior Member, IEEE*, Mehrdad Morsali, Sepehr Tabrizchi, *Student Member, IEEE*, and Arman Roohi, *Senior Member, IEEE*

Abstract—In this work, a high-speed and energy-efficient comparator-based Near-Sensor Local Binary Pattern accelerator architecture (NS-LBP) is proposed to execute a novel local binary pattern deep neural network. First, inspired by recent LBP networks, we design an approximate, hardware-oriented, and multiply-accumulate (MAC)-free network named Ap-LBP for efficient feature extraction, further reducing the computation complexity. Then, we develop NS-LBP as a processing-in-SRAM unit and a parallel in-memory LBP algorithm to process images near the sensor in a cache, remarkably reducing the power consumption of data transmission to an off-chip processor. Our circuit-to-application co-simulation results on MNIST and SVHN datasets demonstrate minor accuracy degradation compared to baseline CNN and LBP-network models, while NS-LBP achieves 1.25 GHz and an energy-efficiency of 37.4 TOPS/W. NS-LBP reduces energy consumption by 2.2× and execution time by a factor of 4× compared to the best recent LBP-based networks.

Index Terms—Processing-in-memory, accelerator, near-sensor processing, SRAM.

1 Introduction

Internet of things' (IoT) nodes consist of sensory systems, which enable massive data collection from the environment and people to process with on-/off-chip processors (10^{18} bytes/s or flops). In most cases, large portions of the captured sensory data are redundant and unstructured. Data conversion and transmission of large raw data to a back-end processor imposes high energy consumption, high latency, and low-speed feature extraction on the edge [1]. To overcome these issues, computing architectures will need to shift from a cloud-centric approach to a thing-centric (data-centric) approach, where the IoT node processes the sensed data. This paves the way for a new smart sensor processing architecture [2], [3], in which the pixel's digital output is accelerated near the sensor leveraging an on-chip processor. Unless a Processing-in-Memory (PIM) mechanism is exploited [4]–[6] in this method, the von-Neumann computing model with separate memory and processing blocks connecting via buses imposes long memory access latency, limited memory bandwidth, and energy-hungry data transfer restricting the edge device's efficiency and working hours [1]. The main idea of PIM is to incorporate logic computation within memory units to process data internally.

From the computation perspective, numerous artificial intelligence applications require intensive multiply-accumulate (MAC) operations, which contribute to over 90% of various deep Convolutional Neural Networks (CNN) operations [5], [7]. Various processing-in-SRAM platforms have been developed in recent literature [5], [8]–[10]. Compute cache [8] supports simple bit-parallel operations (logical and copy) that do not require interaction between bit-lines. Neural Cache [5] presents an 8T transposable SRAM bit-cell and supports bit-serial in-cache MAC operation. Nevertheless, this design imposes a very slow clock frequency and a large cell and Sense Amplifier (SA) area overhead. In [11], a new approach to improve the performance of the Neural Cache has been presented based on 6T SRAM, enabling faster multiplication and addition with a large SA overhead. While the presented designs show acceptable performance over various image datasets by reducing the number of operations, i.e., MACs, using shallower models, quantization, pruning, etc., they are essentially developed to execute the existing CNN algorithms that lead to a gap between meets and needs. We believe such a discrepancy can be avoided by *co-developing an intrinsically-low computation network and an efficient PIM platform on the sensor side*. Regarding the model reduction of CNNs, Local Binary Pattern (LBP)-based implementations have attained worldwide attention for edge devices, resulting in a similar output inference accuracy [12]–[14]. More interestingly, the amount of convolution operations is drastically reduced owing to the sparsity of kernels and conversion to simpler operations such as addition/subtraction [15] and comparison [16].

In this work, inspired by recent LBP networks, (1) we first develop a novel approximate, hardware-oriented, and MAC-free neural network named Ap-LBP in Section 3 to reduce computation complexity and memory access by disregarding the least significant pixels to perform efficient feature extraction. The Ap-LBP is leveraged on the sensor side to simplify LBP layers before even mapping the data into a near-sensor memory; (2) NS-LBP is designed as a comparator-based processing-in-SRAM architecture, in conjunction with the LBP parallel in-memory algorithm in Section 4, which remarkably reduce the power consumption as well as the latency of data transmission to a back-end

This work is supported in part by the National Science Foundation under Grant No. 2228028, 2216772, and 2216773.

- *S. Angizi and M. Morsali are with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA. E-mail: shaahin.angizi@njit.edu.*
- *A. Roohi and S. Tabrizchi are with the School of Computing, University of Nebraska–Lincoln, Lincoln NE, USA. E-mail: aroohi@unl.edu.*

processor; (3) In Section 5, we propose a correlated data partitioning and hardware mapping methodology to process the network locally; and (4) We extensively evaluate NS-LBP performance, energy efficiency, and inference accuracy trade-off compared to recent designs with a bottom-up evaluation framework in Section 6.

2 Background & Motivation

2.1 Near-Sensor & In-Sensor Processing

Systematic integration of computing and sensor arrays has been widely studied to eliminate off-chip data transmission and reduce Analog-to-Digital Converters (ADC) bandwidth by combining CMOS image sensor and processors in one chip as known as Processing-Near-Sensor (PNS) [2], [3], [17]–[19], or even integrating pixels and computation unit so-called Processing-In-Sensor (PIS) [20]–[26]. However, since enhancing the throughput and increasing the computation load on the resource-limited IoT devices is followed by a growth in the temperature and power consumption as well as noises that lead to accuracy degradation [20], [27], the computational capabilities of PNS/PIS platforms have been limited to less complex applications [1], [28]. This includes particular feature extraction tasks, e.g., Haar-like image filtering [28] and blurring [3].

Various powerful processing-in-SRAM (in-cache computing) accelerators have been developed in recent literature that can be employed as a PNS unit [5], [8]–[11], [29]–[32]. XNOR-SRAM [10] accelerates ternary-XNOR-and-accumulate operations in binary/ternary Deep Neural Networks (DNNs) without row-by-row data access. C3SRAM [9] leverages capacitive-coupling computing to perform XNOR-and-accumulate operations for binary DNNs. However, both XNOR-SRAM and C3SRAM impose huge overhead over the traditional SRAM array by directly modifying the bit-cells. In [11], a new approach to improve the performance of the Neural Cache has been presented based on 6T SRAM, enabling faster multiplication and addition with a large SA overhead. In the PIS domain, a CMOS image sensor with dual-mode delta-sigma ADCs is designed in [33] to process 1^{st}-convolutional layer of Binarized-Weight Neural Networks (BWNNs). RedEye [34] executes the convolution operation using charge-sharing tunable capacitors. This design reduces energy consumption compared to a CPU/GPU by sacrificing accuracy. However, to achieve high-accuracy computation, the required energy per frame increases dramatically by 100×. The presented in-SRAM computing macro in [35] has been fabricated in 28nm process technology and works based on approximate arithmetic hardware that negatively affects CNN accuracy (25.2% on CIFAR-10). To improve the accuracy, the authors use approximation-aware training and a new number format called multi-bit XNOR. Macsen [20] processes the 1^{st}-convolutional layer of BWNNs with the correlated double sampling procedure achieving 1000fps speed in computation mode. However, it suffers from humongous area-overhead and power consumption.

There are **three main bottlenecks** in IoT imaging systems that this paper aims to solve: (1) The data access and movement consume most of the power ($> 90\%$ [20], [29]) in conventional image sensors; (2) the computation imposes a large area-overhead and power consumption in more recent PNS/PIS units and requires extra memory for intermediate data storage; and (3) the system is hardwired so their performance is intrinsically limited to one specific type of algorithm or application domain, which means that such accelerators cannot keep pace with rapidly evolving software algorithms.

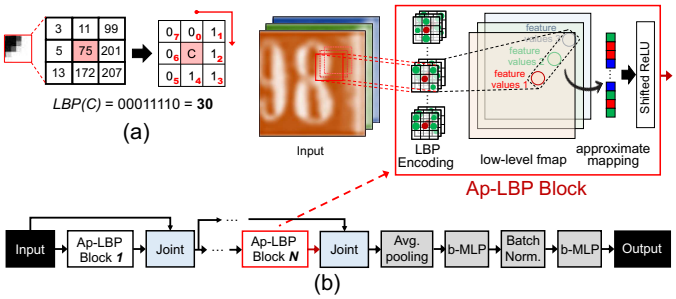

Figure 1: (a) Standard LBP encoding with 3×3 descriptor size, (b) The structure of the Ap-LBP, with N Ap-LBP blocks.

2.2 LBP-based Networks

An LBP kernel is a computationally efficient feature descriptor that scans through the entire image like that of a convolutional layer in a CNN. The LBP descriptor is formed by comparing the intensity of surrounding pixels serially with the central pixel, referred to as Pivot, in the selected image patch. Neighbors with higher (/lower) intensities are assigned with a binary value of '1'(/'0') and finally, the bit stream is sequentially read and mapped to a decimal number as the feature value assigned to the central pixel, as shown in Fig. 1(a). The LBP encoding operation of central pixel $C(x_c, y_c)$ and its reformulated expression can be mathematically described as $LBP(C) = \sum_{n=0}^{d-2} cmp(i_n, i_c) \times 2^n$ [12], where d is the dimension of the LBP, i_n and i_c represent the intensity of n^{th} neighboring- and central-pixel, respectively; thus, $cmp(i_n, i_c) = 1$ when $i_n \geq i_c$, otherwise outputs 0. Simulating LBP is accomplished using a ReLU layer and the difference between pixel values.

The Local Binary Pattern Network (LBPNet) [36] and Local Binary Convolutional Neural Network (LBCNN) [15] are two recent LBP networks where the convolutions are approximated by local binary additions/subtractions and local binary comparisons, respectively. It should be noted that LBPNet and LBCNN are quite different, despite their similarity in their names, as illustrated in Fig. 2. In the LBCNN, batch norm layers are still heavily utilized, which are completed in floating-point numbers for the linear transform. Moreover, since the size and computation of 2D batch norm layers are linear in the size of the feature maps, model complexity increases dramatically. Therefore, the use of LBCNNs for resource-constrained edge devices, such as sensors, is still challenging and impractical. LBPNets, on the other hand, learn directly about the sparse and discrete LBP kernels, which are typically as small as a few KBs. By using LBPNet, the computation of dot products and sliding windows for convolution can be avoided. Rather, the input is sampled, compared, and then the results of the comparisons are stored in determined locations. A local binary comparison and random projection are used instead of conventional convolutions. An output channel is selected

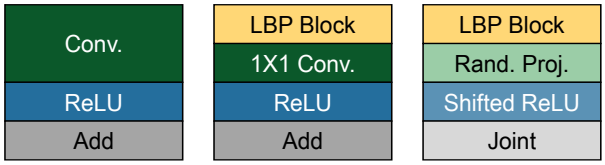

Figure 2: Different basic building blocks of a (a) residual network, (b) LBCNN [15], and (c) LBPNet [36].

from intermediate channels using the random projection layer as a dimension-reducing process. Therefore, in LBP-Net, only trained patterns of sampling locations are held, and no MAC operations (convolution-free) are performed, making it a hardware-friendly and suitable model for edge computing.

3 Ap-LBP Network

The Ap-LBP network is trained similarly to the LBPNet, which learns a set of local binary patterns. The Ap-LBP structure, visualized in Fig. 1(b), consists of multiple LBP layers followed by an average pooling, two Multi-Layer Perceptron blocks (MLP), and one batch normalization layer. A standard convolutional layer is replaced with a layer using LBPs, which means neither multiplication nor addition is required, and MAC operations are performed via memory access and comparison. An LBP layer, including an LBP Block and a Joint operation, is leveraged to extract feature maps. Each LBP block consists of an LBP Encoding step that can be readily implemented by a comparator[1] to generate new feature maps connected to an approximate mapping and shifted-ReLU blocks to increase nonlinearity. The output of the LBP blocks is cascaded with the input feature maps (ifmaps) using joint blocks. Figure 3 illustrates a portion of the LBP block's operation. In the Ap-LBP, the size of the output feature maps (ofmaps) remains identical to the size of the ifmaps. To do so, the zero-padding approach might be utilized and the degree of zero insertions is calculated by $pad = [s \times (out-1) - in + f]/2$, where s is the stride window's size, *out* and *in* are sizes of the ofmap and the ifmap, respectively, and f is the size of the LBP kernel. This expression works for square matrices. For example, as shown in Fig. 3(a) with s=1, *in*= 5 and f= 3, to produce an ofmap with *out*= 5, zero-padding approach with a degree of one should be utilized.

The learned sets of LBPs from the training step are used in the encoding part to denote the sampling points in ifmaps' positions that are to be compared with a pivot. After the training phase, pre-defined locations in encoding matrices and bit arrays are determined and remain fixed during the inference phase, e.g., LBP Kernels 1 and 2 in Fig. 3(b). Since the given weights to the thresholded (compared) values and mapping patterns are specified, a *Partial Approximate Computing method (PAC)* is developed to further improve the performance at the cost of lower accuracy. The PAC includes two primary operations: (1) Skip comparison:

1. In the backward propagation, binary comparisons are replaced by a modified hyperbolic tangent (tanh) function and shifted to become differentiable.

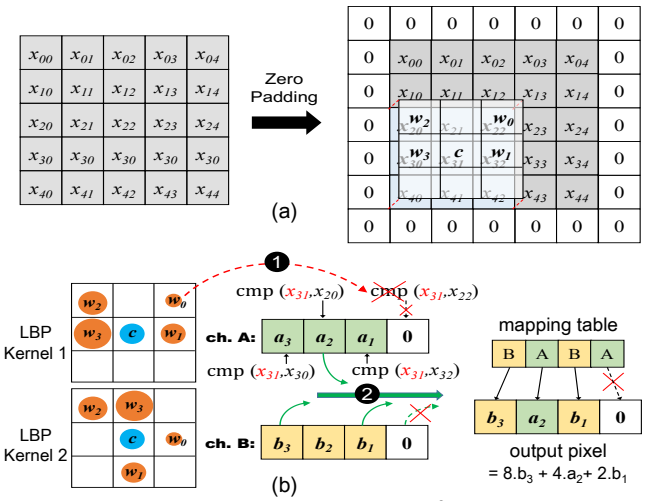

Figure 3: (a) Visualization of X and X^{p0}, zero padding; the blue boxes is the sliding 3×3 LBP kernel, (b) Approximate mapping using comparison and memory access skipping schemes on channels **A** and **B** to generate ofmap.

since in the LBP layer, the positions (weights) of LBP kernels' elements are already specified, pixel-to-pivot comparison operation related to the Least Significant Bits (LSB) can be omitted, and ofmap is written by zero, step ❶ shown in Fig. 3(b). (2) Skip memory access: in the LBP channel fusion step, a pre-defined mapping table, referred to as a projection map, is fixed for all outputs within the same channel to generate an output pixel passing through the shifted-ReLU function. Accordingly, read (/write) operations from (/to) the LSBs of the channel's responses (output pixels) can be skipped, as shown in step ❷ shown in Fig. 3(b). By leveraging the PAC, comparison operations and memory accesses can be reduced as significant portions of Ap-LBP computation in order to minimize energy consumption with minimum accuracy loss. For example, in Fig 3(b), the original LBPNet implementation requires 8 comparisons, 14 read and 12 write operations; however, using Ap-LBP, the output pixel can be generated by 6, 11, and 9 comparisons, read and write operations, respectively, which shows a considerable enhancement. In other words if the output pixel is *n-bit*, for each output pixel, there will be n comparisons needed, which is irrelevant to the LBP dimension and the number of input channels. Random projection tables can be constructed using more input channels with higher resolutions, resulting in more combinations of representations. But the spatial dimension of output pixels affects the final results. The total number of operations required to produce output pixels in the LBP blocks, utilizing LBPNet and Ap-LBP can be computed by the following expressions:

$$OP_{LBPNet} = \underbrace{[e \times ch + m]}_{\#read} + \underbrace{[(e-1) \times ch]}_{\#comparison} + \underbrace{[(e-1) \times ch + m]}_{\#write} \quad (1)$$

$$OP_{Ap\_LBP} = \underbrace{[(e - apx) \times ch + m - apx]}_{\#read} + \underbrace{[(e - apx - 1) \times ch]}_{\#comparison} + \underbrace{[(e - apx - 1) \times ch + m - apx]}_{\#write}, \quad (2)$$

where e is the number of LBP kernels' elements (number of samplings), *ch* is the number of channels, m is the number

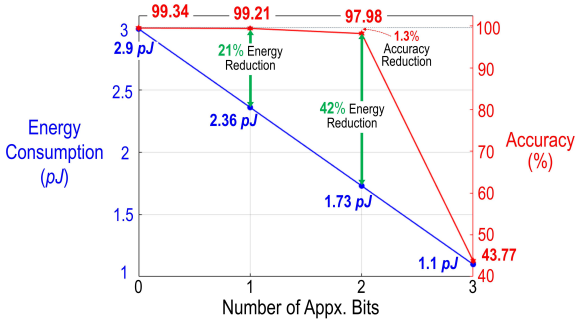

Figure 4: Energy consumption vs. accuracy regarding the number of approximated bits on MNIST dataset.

of mapping tables' elements, and *apx* is the number of approximated bits. Increasing *apx* results in higher speed and energy efficiency at the cost of accuracy degradation. Figure 4 illustrates the accuracy results for Ap-LBP on MNIST with respect to the number of approximated bits and energy consumption of the LBP layers. This figure shows trade-offs between energy consumption and accuracy with respect to the number of bits in our design. The results are achieved based on the framework setup that will be introduced in Section 6.1. The left axis (blue) contains hardware implementation results for the Ap-LBP processing of the MNIST dataset and the right axis (red) is achieved by our software-layer analysis based on the Pytorch model. As can be seen, the optimal condition occurs when 2 of 4 mapping tables' bits are approximated, which leads to relatively high energy savings (42%) despite a small reduction in accuracy (1.3%). In addition, the computational and memory costs for the convolution layer of both conventional CNNs and Ap-LBP networks are presented in Table 1. As shown, in the convolutional layer, the dimension of filters (ifmaps) is 4-D, $K \times ch \times r \times s$ ($M \times ch \times h \times w$), where K and M are the number of filters and ifmap, respectively, ch is the number of channels, $r \times s$ is the spatial dimension of filters and $h \times w$ is the dimension of 2-D ifmaps. So the generated ofmaps' dimensions are $M \times K \times p \times q$, where $p \times q$ is ofmap's 2-D dimensions. To simplify matters, a single kernel ($K = 1$) and a single ifmap ($M = 1$) are considered. Since the difference between the number of samplings in an LBP pattern, e, and the number of approximated bits, apx, is relatively smaller than the spatial dimensions of kernels, that Ap-LBP, with MAC-free LBP layers, significantly reduces the hardware cost, both computation, and memory.

Table 1: Hardware cost analysis of CNN vs. Ap-LBP.

Network	Computational cost		Memory cost
	Mul–$O(N^2)$	Add/Sub/Cmp–$O(N)$	
CNN	$p \cdot q \cdot ch \cdot r \cdot s$	$p \cdot q \cdot ch \cdot r \cdot s$	$p \cdot q \cdot r \cdot s$
Ap-LBP	-	$ch \cdot p \cdot q \cdot (e - apx)$	$p \cdot q \cdot (e - apx) + (m - apx)$
$\frac{Ap-LBP}{CNN}$	0	$\frac{(e-apx)}{r \cdot s}$	$\frac{(e-apx)}{r \cdot s} + \frac{(m-apx)}{p \cdot q \cdot r \cdot s}$

4 Proposed NS-LBP

4.1 Architecture

We propose NS-LBP as a cache-based near-sensor architecture to accelerate the Ap-LBP network with a parallel in-memory LBP algorithm. NS-LBP is mainly developed to process Ap-LBP's key operations. LBP layers are accelerated through an efficient comparison implemented with data-parallel X(N)OR bit-wise operations and the MLP layers are executed near-sensor through data-parallel AND-bit count operations as explained below. However, NS-LBP can be used to accelerate 2- and 3-input bulk bit-wise operations in various applications such as data encryption, graph processing, etc. NS-LBP's geometry of a single 2.5MB cache connected to an image sensor is shown in Fig. 5(a). A rolling-shutter CMOS image sensor is composed of m×n photodiode-based pixels, which utilize the Correlated Double Sampling (CDS) mechanism [20]. CDS measures the photodiode's voltage drop before and after an image light exposure and utilizes an ADC to convert it to a digital value. However, a significant amount of power is consumed by ADC conversion of raw images and high-throughput transmission [1], [3], [20].

To reduce the power consumption imposed by ADC and data transmission to the memory, we first modify the sensor controller and peripheral circuitry so that Ap-LBP's approximation can be applied on the sensor side by simply avoiding pixel conversion for less significant bits. This is explained in Section 3. By using this mechanism, the NS-LBP is assured of receiving only compute pixels and pivots. Cache slices within NS-LBPs are designed to have 80 memory banks, of 32KB each organized in 20 distinct ways. Each bank contains two 16KB memory matrices-mat (see Fig. 5(a)). The centralized control unit (Ctrl) manages the internal memory data transfer, intra-bank computation, and a digital processing unit (DPU) common to all memory banks. The main computational cores of NS-LBP are 8KB computational sub-arrays as depicted in Fig. 5(b)-(c).

According to our observations of existing sub-array-level processing-in-SRAM platforms, they face various challenges, such as multi-cycle in-memory operations, word-line underdrives, high-latency, read disturbances, etc. [11], [30], [32], when it comes to comparison and addition operations required by the proposed Ap-LBP. The proposed NS-LBP's sub-array (Fig. 5(c)) leverages the voltage discharging profile of the read-write-decoupled 8T SRAM cell (Fig. 5(d)) on Read-BL (RBL) used for the standard read operation and elevates it to implement Boolean logic between operands located in different memory rows in a single SRAM read cycle. In this way, we develop a processing-in-SRAM sub-array through a three-row activation mechanism by modifying the memory row decoder, SA, and Ctrl. It is important to note that the key idea comes from the observation that certain discharge rates on the precharged RBL can be expected based on selected memory bits. For instance, by activating three memory rows via Read Word-Lines (RWL), e.g., RWL0-RWL2 shown in Fig. 5(c), if $S_{0,0}$, $S_{1,0}$, and $S_{2,0}$ memory cells hold binary "1", then the read access transistors (T8 in Fig. 5(d)) remain OFF, and the RBL precharged voltage doesn't degrade. However, if all cells hold binary "0", the RBL voltage is rapidly discharged through T8s. Accordingly, we propose a new reconfigurable SA as shown in Fig. 5(e) consisting of three sub-SAs, each dedicated to computing a particular function.

With a proper selection of a reference voltage ($R_1 < R_2 < R_3$), each sub-SA performs a neat voltage comparison with RBL voltage and generates (N)OR3, (MAJ)MIN, and (N)AND3 logic functions simultaneously. The XOR-based

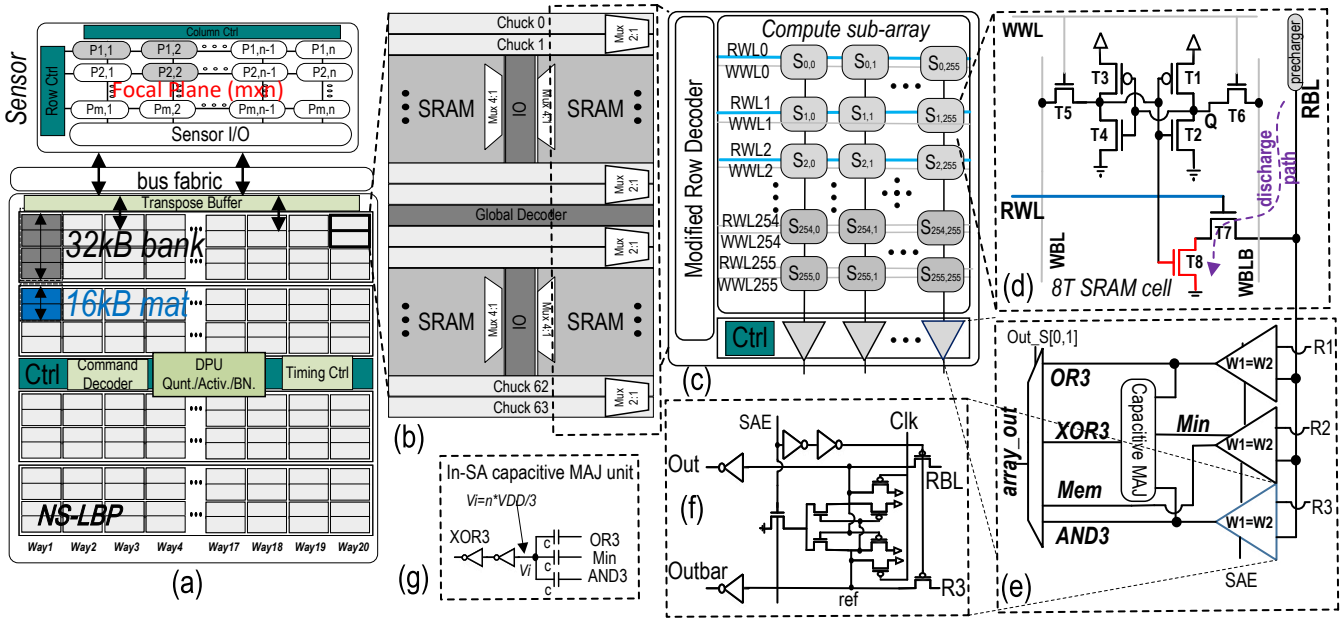

Figure 5: (a) The NS-LBP's geometry of a single 2.5MB cache slice connected to a sensor, (b) Computational matrix, (c) An 8KB SRAM computational sub-array, (d) 8T SRAM cell, (e) Proposed single-cycle SA design for LBP extraction, (f) Inside SA unit, (g) Capacitive majority function generator circuit.

Table 2: NS-LBP ISA.

Opcode	Src1	Src2	Src3	Dest	Size	Description
NS-LBP copy	r1	-	-	r2	n	r2[i] = r1[i]
NS-LBP ini	r1	-	a[i] = all '0' b[i] = all '1'	-	n	r1[i] = a[i] or r1[i] = b[i]
NS-LBP cmp (xor2)	r1	r2	a[i] = all '0'	r3	n	r3[i] = r1[i] ⊕ r2[i]
NS-LBP search	r1	k	-	r3	n	r3[i]= (r1[i] == k)
NS-LBP nand3	r1	r2	r3	r4	n	r4[i]= !(r1[i] & r2[i] & r3[i])
NS-LBP nor3	r1	r2	r3	r4	n	r4[i] = !(r1[i] ‖r2[i]‖r3[i])
NS-LBP carry (maj3)	r1	r2	r3	r4	n	r4[i] = maj(r1[i], r2[i], r3[i])
NS-LBP sum (xor3)	r1	r2	r3	r4	n	r4[i] = r1[i] ⊕ r2[i] ⊕ r3[i]
r1-r4: addresses			k:address	$\forall i, i \in [1, n], X = [64/128/256]$		

comparison is then achieved through an observation in which the three input majority function of OR3, MIN, and AND3 is able to generate XOR3 logic. The Boolean logic of in-memory XOR3 can be given as $XOR3(/Sum) = MAJ((A + B + C) + (\overline{AB + AC + BC}) + (ABC))$. This unit is implemented with a low overhead capacitive voltage divider as shown in Fig. 5(g). The implementation of 2-input bit-wise operations is straightforward by initializing one row to "0"/"1". We choose an 8T SRAM cell as a fast and compact design considering that the proposed in-memory computing mechanism operates based on BL discharging. Nevertheless, the mechanism presented here can be applied to various read-write decouple SRAM designs.

From a programmer's perspective, NS-LBP is interfaced as a bus-facing accelerator that can be connected directly to the memory bus or through PCI-Express lanes rather than a memory unit. Therefore, a virtual machine and ISA for general-purpose parallel thread execution need to be defined. We designed instruction sets that could optimally leverage highly parallel NS-LBP's operations discussed and developed a compilation framework on top of that. Accordingly, the programs will be translated at install time to the NS-LBP's hardware ISA tabulated in Table 2.

4.2 In-memory LBP Algorithm

By converting a conventional software-based sequential comparison operation into a parallel bit-wise XOR operation, we propose an NS-LBP hardware-oriented LBP algorithm that fully utilizes the sub-array parallelism of the NS-LBP. A key objective in developing such an algorithm is to enable a parallel bit-position-aware comparison between pivot (C) and surrounding pixels (P) and generate an LBP bit-stream in fewer cycles, eliminating unnecessary power-hungry bulk bit-wise operations. For every LBP kernel, starting from the Most Significant Bit (MSB), Algorithm 1 issues the NS-LBP's comparison command (NS-LBP_XOR) in a loop to pivot and pixels in parallel and update the Result_array (line-7). The result of i^{th} bit comparison ($C_i \oplus P_{j,i}$) is leveraged as a determining factor for NS-LBP to take the next step. As indicated in the algorithm, when the XOR result is "1", i.e., two unequal bits are identified (line-8), C_i is read (NS-LBP_Mem). Now, if C_i equals "0", the corresponding LBP_array position is set by "1", indicating $C_i < P_j$ and vice versa (lines-9-12). However, if equality is noticed, the next less significant bit in pixels and pivot is selected for comparison, and this process stops when the XOR result is "1" (inequality). Such a parallel comparison operation could rapidly detect the mismatch between all pixels and pivot from MSB to LSB. Our algorithm has a constant search time that is determined by the bit length of numbers. As shown, NS-LBP_XOR is iteratively used in a nested "for" loop in the algorithm, and the NS-LBP architecture is mainly designed to accelerate this operation.

5 Correlated Hardware Mapping

5.1 LBP Layer

To maximize Ap-LBP computation throughput and fully leverage NS-LBP's parallelism, we propose partitioning data as shown in Fig. 6. Given an LBP layer, the accessed memory region of pixels and pivots could be easily predicted, and the LBP bit-stream could be locally computed if we could store such correlated regions into the same memory sub-array. Thus, we propose a novel, correlated data partitioning, and mapping methodology as shown in Fig. 6(a) to locally store correlated regions of pixel

Algorithm 1 Parallel Bit-wise In-memory LBP Algorithm

Input: Pixel 2D-Array (P[][]) with M=8 elements (P_7 to P_0), where each element contains N=8 bits. Pivot 1D-Array (C[]), LBP_array=0.
Output: Returning the LBP bit stream value in the given array (LBP_array).
procedure `NS-LBP(P[][],C[],LBP_ARRAY)`
  Result_array$\leftarrow 0$
  for $i \leftarrow N-1$ to 0 **do**
    for $j \leftarrow N-1$ to 0 **do**
      Result_array $\leftarrow$ `NS-LBP_XOR`$(C_i, P_{j,i})$ ▷ Bit-position compare from MSB to LSB.
      if Result_array$_j$==1 **then**
        if `NS-LBP_Mem`$(C_i) == 0$ **then**
          LBP_array$_j \leftarrow 1$ ▷ $C < P_j$
        else
          LBP_array$_j \leftarrow 0$ ▷ $C > P_j$
      else
        break ▷ Continue with the lower SB compare
  return LBP_array
end procedure

and pivot vectors in the same memory sub-array and enable entirely local computation (i.e., NS-LBP_XOR and NS-LBP_Mem completely within the same sub-array without inter-bank/chip communication). The NS-LBP's compute sub-array (256 rows×256 columns) is split into five key regions, i.e., Pixel-P (64 rows), Pivot-C (64 rows), Reserved (64 rows), Weight-W (32 rows), and Input-I (32 rows). We use P-, C-, and Resv. regions to process the LBP layer.

The selected input pixels in Ap-LBP are initially transposed in the NS-LBP's buffer and mapped into the P-region. In addition, we propose to store P_{i+1} transposed copy of the pivot as reference vectors in the C-region. The C-region is specially designed to enable fully parallel bit-wise position-aware comparison operation. Three rows in the Resv. region are dedicated to Result_array, LBP_array, and all-zero. Figure 6(b) gives an intuitive example of LBP-layer computation with the in-memory LBP algorithm, where four pixels ($P3$ to P_0) are selected. After data mapping, NS-LBP's Ctrl activates three RWLs simultaneously, corresponding to pixels' and pivot's MSB and all-zero row. The NS-LBP

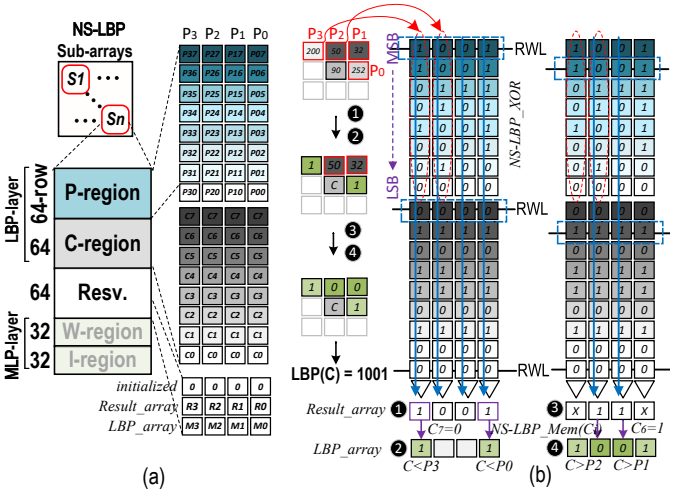

Figure 6: (a) The NS-LBP's correlated data partitioning and mapping scheme, (b) Parallel LBP computation in NS-LBP.

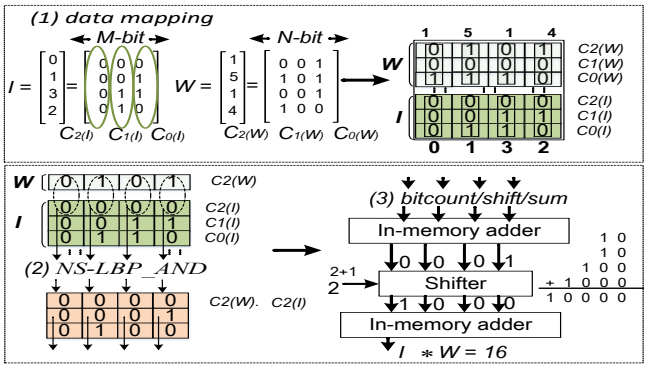

Figure 7: Parallel MLP computation in NS-LBP.

sub-array then performs the parallel XOR2 operation in a single cycle based on the mechanism discussed in Section 4, and the result "1001" is stored in the Result_array row (step ❶). Now, the Ctrl readily recognizes the potential mismatch in P_3 and P_0 and accordingly updates LBP_array in step ❷ ("1xx1") with respect to C_7=0 value. As there are two matches ($P_{2,7}$-C_7 and $P_{1,7}$-C_7), the Ctrl selects the next MSBs in pixels and pivot to find the next potential mismatch. The final LBP_array value ("1001") is returned in step ❹ for the next step. It is worth pointing out that other configurations of SRAM-based cache memory can be readily adopted and used. The only constraint is to assure the converted sensor data can be properly stored to support the NS-LBP correlated data partitioning and mapping scheme.

5.2 MLP Layer

Besides the LBP layer, there are MLP layers in Ap-LBP as shown in Fig. 1(b) that can be accelerated close to the sensor without sending the activated LBP feature maps to an off-chip processor. Note that MLP can be equivalently implemented by convolution operations using 1×1 kernels [37]. W- and I- regions in every NS-LBP sub-array (Fig. 6(a)) are dedicated to performing such an operation locally. Figure 7 gives an overview of the MLP bit-wise acceleration steps. In the first step, the processed input activation from NS-LBP's LBP layers is quantized by DPU and mapped into I-region, where the MLP layer weights are located. In the second step, parallel computational sub-arrays perform bulk bit-wise operations between tensors and generate the output. Then, the output is activated by DPU's Activation unit and saved back into the Resv. region. From a computation perspective, every MLP layer can be equivalently implemented by exploiting NS-LBP_AND, bitcount, and bitshift as parallelizable operations [37].

Assume I is a sequence of M-bit input integers, e.g., 3-bit in Fig. 7 located in ifmap covered by a sliding kernel of W, such that $I_i \in I$ is an M-bit vector representing a fixed-point integer. We index the bits of each I_i element from LSB to MSB with $m = [0, M-1]$, such that $m = 0$ and $m = M-1$ are corresponding to LSB and MSB, respectively. Accordingly, we represent a second sequence denoted as $C_m(I)$ including the combination of m^{th} bit of all I_i elements (shown by colored elliptic). For instance, $C_0(I)$ vector consists of LSBs of all I_i elements "0110".

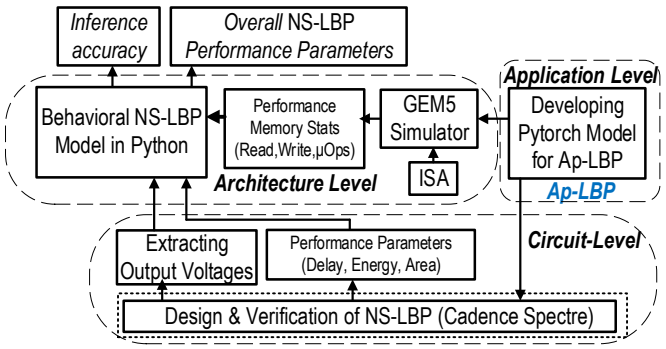

Figure 8: Evaluation framework developed for NS-LBP accelerator.

Considering W as a sequence of N-bit weight integers (3-bit, herein) located in a sliding kernel with an index of $n = [0, N-1]$. The second sequence can be similarly generated as $C_n(W)$. Considering the set of all m^{th} value sequences, the I can be represented like $I = \sum_{m=0}^{M-1} 2^m c_m(I)$. Likewise, W is represented like $W = \sum_{n=0}^{N-1} 2^n c_n(W)$. Thus, the convolution between I and W is defined as $\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} 2^{m+n} bitcount(and(C_n(W), C_m(I)))$ [37]. In the data mapping step of Fig. 7, $C_2(W)$-$C_0(W)$ and $C_2(I) - C_0(I)$ are consequently mapped into an NS-LBP's sub-array. Now, a parallel bit-wise AND operation (NS-LBP_AND) of $C_n(W)$ and $C_m(I)$ is performed. The results will be then processed using a bit-counter counting the number of "1"s in each vector and then a shifter unit, e.g., here left-shifted by 3-bit ($\times 2^{2+1}$) to "1000". Eventually, the shifter unit's outputs are added up to produce ofmaps for every layer.

6 Evaluation Results

6.1 Setup

To estimate the performance of NS-LBP along with Ap-LBP, a bottom-up evaluation framework is developed as shown in Fig. 8. At the *circuit level*, NS-LBP is fully implemented with TSMC 65nm-GP with a supply voltage of 0.9V-1.1V in Cadence, and the post-layout results are reported. However, NS-LBP is not taped out. The NS-LBP platform can be readily implemented in lower technology nodes to achieve lower power consumption and higher TOPS/W. At the *architecture level*, we fully implemented NS-LBP's ISA using gem5 [38]. The compiler is then developed on top of the PIMSim's full-system mode [39] taking array parameters (latency and energy consumption for individual operations) and the binary of the application as input and exporting the memory statistics and performance evaluation results. The results are then fed into a behavioral NS-LBP's in-house optimizer tool, also taking the circuit-level data to model the timing, energy, and area. This tool will offer the same flexibility in memory configuration regarding bank/mat/sub-array organization and peripheral circuitry design as Cacti [40] while supporting SRAM-level configurations. The architecture simulator can alter the configuration files with different array organizations. At the *application level*, we trained a PyTorch implementation of Ap-LBP inspired by LBPNet, with the difference that our design approximates pre-trained LBP kernel parameters. The Ap-LBP's statistics are then leveraged in the behavioral NS-LBP model to compute the latency and energy of the whole system. Besides,

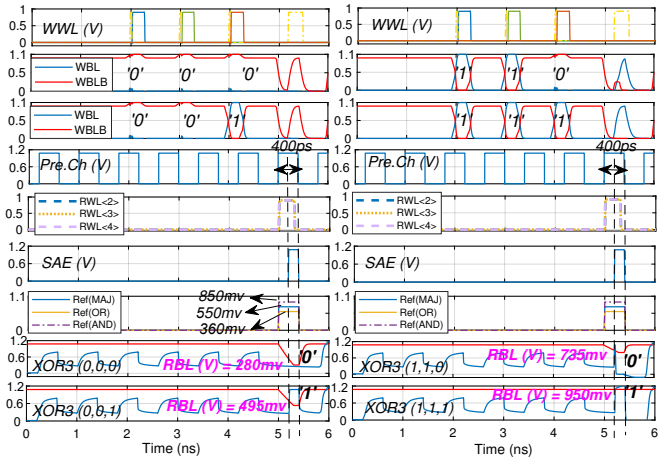

Figure 9: Transient simulation results of an NS-LBP sub-array executing comparison (based on XOR3) operation.

to model the data loading time for all layers, we followed the approach in [5] by developing a micro-benchmark that sequentially accesses the sets in a way that requires data loading. In fact, the Ap-LBP network is the result of two sets of experiments. First, a software (PyTorch) implementation was accomplished to analyze the final network accuracy with various approximation methods including the PAC method. This experiment clearly showed us the expected accuracy degradation. Second, the Ap-LBP is fully implemented in our digital in-memory accelerator to achieve the expected accuracy.

6.2 Functionality Analysis

Figure 9 shows the post-layout transient simulation results of an NS-LBP sub-array. To verify the functionality of all possible input combinations ("000", "001", "011", and "111"), three WWLs are activated consecutively (first waveform) and by assigning proper voltages to WBL and WBLB, the SRAM cells are loaded with the operands. In the computation mode, we simultaneously activate the corresponding RWL of three cells to discharge RBL from the precharged voltage (1.1V) w.r.t. the memory value. To compromise three-row activation stability by lowering the RWL voltage that leads to read latency, we reduced the RWL voltage to 790mV to achieve the industry standard 6-sigma margin. For the evaluation, by activating the Sense Amplifier Enable (SAE) signal, a voltage comparison between the RBL voltage and references is made. As shown in Fig. 9, V_{R1}=360mV, V_{R2}=550mV, and V_{R3}=850mV are set as the reference voltages.

In the case of "000", T8s (see Fig. 5(d)) of all three cells are ON pulling down the RBL voltage from 1.1V to 280mV. This can be easily detected by SA generating "0" as the XOR3 output ($V_{R3} > V_{R2} > V_{R1}$ >280mV). The total processing time from enabling the SA to get the result is ~400ps in the same range given by the standard foundry memory compiler. In the case of "001", T8s of two out of three cells are ON pulling down the RBL voltage from 1.1V to 495mV. This can be easily detected by SA generating "1" as the XOR3 output ($V_{R3} > V_{R2}$ >495mV> V_{R1}). With "011", T8 of only one cell is ON pulling down the RBL

voltage from 1.1V to 735mV generating "0" as the XOR3 output (V_{R3} >735mV> V_{R2} > V_{R1}). Eventually, with "111" as inputs, all T8s are OFF taking the RBL voltage at 950mV outputting "1" (950mV> V_{R3} > V_{R2} > V_{R1}).

For the SA reference voltage (V_R) analysis, the RBL sense margins are first tested through post-layout Monte Carlo simulations in Cadence Spectre, as shown in Fig. 10, where the sensing margin is reported considering both process (inter-die) and mismatch variations (intra-die) for core VDD (1.1 V) at 1.25 GHz. To conduct the V_R variation analysis, we tested all 256 bit-lines within each NS-LBP's sub-array, 200 times, for all possible bit value combinations in memory. It is found that at lower voltages the maximum operating frequency is limited by the reduction of V_R ranges. A higher VDD also yields a larger sensing margin. As we observe there is ∼92mV margin (the smallest voltage margin observed between "111" and "011" cases) between every two combinations bringing high in-memory computing reliability for the NS-LBP design.

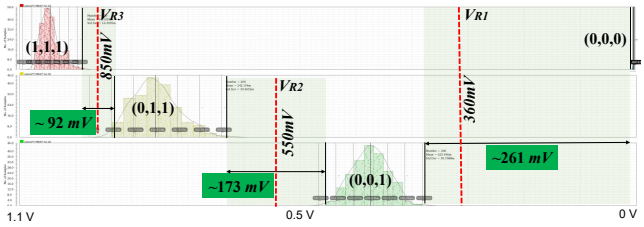

Figure 10: Monte-Carlo simulation of RBL and SA reference voltage.

6.3 Energy Consumption & Performance

Figure 11(a) shows the energy consumption breakdown of NS-LBP running Ap-LBP and LBPNet compared to a baseline 8-bit quantized CNN and LBCNN implemented by [30] running SVHN dataset. We meticulously report the energy consumed by MAC and CMP operations in various networks. We observe that (i) the NS-LBP running Ap-LBP demonstrates up to ∼2.2× and 5.2× higher energy efficiency compared to the LBPNet and CNN counterparts, respectively. Converting power-hungry MAC to bit-wise comparison operation in an approximate fashion has yielded such a striking improvement; (ii) leveraging Ap-LBP can bring up to ∼4× energy-efficiency when compared with the LBCNN. It is worth mentioning that LBCNN still relies on power-hungry MAC operations. Figure 11(b) compares inference delay per input image in four under-test designs. We observe that the NS-LBP leveraging Ap-LBP achieves ∼4× and 2.3× speed-up compared to LBPNet and LBCNN designs, respectively. Besides, it can be seen that ∼6.2× speed-up is achieved when compared with the CNN baseline. Figure 11(c) further clarifies that Ap-LBP doesn't remarkably reduce the memory storage relative to LBP-Net; however, it requires ∼3.4× smaller memory to store the parameters than LBCNN.

6.4 Comparison

Since several processing-in-SRAM platforms have been developed to accelerate various deep neural networks in literature, performing a fair comparison is a difficult task. Nevertheless, Table 3 lists seven recent designs for a comparison.

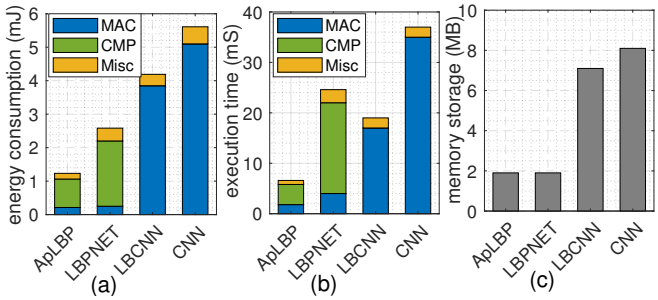

Figure 11: (a) Energy consumption, (b) Execution time, and (c) Memory storage comparison.

We compared our digital approximate LBP accelerator with conventional analog/digital MAC-based neural network accelerators supporting bit-truncation (/quantization) as a well-known method in neural network approximation. As can be seen, various designs are implemented with different bit-cell structures and SA designs. Here we report our main observations. (1) The NS-LBP and the designs in [30], [31] are the only in-SRAM platforms that can support XOR-based LBP computation. We observe that NS-LBP shows a fairly smaller SA area overhead (3.4×) compared to these designs to support in-memory computation. (2) It can be seen that the designs presented in [11], [31] show the highest frequency at 1V, where NS-LBP stands as the third-fastest design. (3) The NS-LBP achieves 37.4 TOPS/W standing as the fourth most efficient design compared to all counterparts, whereas the design in [9] with 671 TOPS/W stands as the most efficient design. To assess the impact of the technology node on the overall performance, we implemented the NS-LBP with 22nm Predictive Technology Model (PTM) library [42] and extracted the pre-layout results. Based on our observations, a higher performance (87.3 TOPS/W) and lower power consumption can be achieved.

Overall, NS-LBP offers 1) A dual-mode computational SRAM platform with no sacrifice of memory capacity that directly processes data within the memory array to eliminate off-chip data communication; 2) A complete set of Boolean operations (both 2- and 3-input), majority, and full adder in only one single memory cycle, demonstrating one of the most efficient PNS systems to date; 3) Highly parallel low-bit-width convolution operation; and 4) Light modification of existing memory cell to achieve low in-memory logic area overhead.

6.5 Accuracy

To perform a fair comparison between Ap-LBP and five other neural network models, CNN (as a baseline) [43], Binarized Neural Network (BNN) [44], BinaryConnect [45], LBCNN [15], and LBPNet [36], with identical hyper-parameters such as number of basic blocks, number of hidden neurons, etc. are selected. We conduct experiments on five datasets, i.e., MNIST, FashionMNIST, SVHN, CIFAR-10, and CIFAR-100 to evaluate the performance of both algorithm accuracy and hardware implementation. PyTorch implementation of Ap-LBP inspired by LBPNet, with the difference that our design approximates pre-trained LBP kernel parameters, is developed. The number of basic blocks for MNIST and SVHN is set to 5 (3 LBP layers and 2 FC layers) and 10 (8 LBP layers and 2 FC layers) layers,

Table 3: Comparison with previous processing-in-SRAM accelerators.

Reference	NS-LBP	Symp. VLSI [41]	DAC'20 [11]	JSCC'20 [9]	JSSC'19 [30]	DAC'19 [31]	ISSCC'19 [32]	ISSCC'22 [35]
Technology	65nm	65nm	28nm	65nm	28nm	28nm	28nm	28nm
Bitcell Density	8T	10T1C	6T	8T-1C	8T Transposable	6T/local group	8T	8T
SA compute Area Overhead	3.4×	-	4.94×	-	5.52×	5.05×	>15×	-
LBP-comparison Support	Yes	No	No	No	Yes	Yes	No	No
MAC Support	Yes (digital CNN)	Yes (analog BWNN)	Yes (digital CNN)	Yes (analog BWNN)	Yes (digital CNN)	No	Yes (analog BWNN)	Yes (digital BNN)
Supply	0.9V-1.1V	0.68-1.2V	0.6V-1.1V	0.6V-1V	0.6V–1.1V	0.6V –1.1V	0.6-0.9V	0.45-1.1
Max Frequency	1.25GHz (1.1V)	100MHz	2.25GHz (1V)	50MHz	475MHz (1.1V)	2.2GHz (1.0V)	400MHz	250MHz
TOPS/W	37.4	658	8.09 (0.6V, 372MHz)	671.5	5.27 (0.6V, 114MHz)	-	5.83	154-248
Array size	4×256× 256	-	4×128× 128	4×128× 128	4×128×256	256×64	4*I: 28x28x4x6x8/ 4*W: 28x28x4x2	256× 64

respectively, with 512 hidden neurons. As for CIFAR-10 and CIFAR-100 datasets, the number of basic blocks is respectively set to 7 (5 LBP layers and 2 FC layers) and 17 (14 LBP layers and 3 FC layers) layers, respectively, with 512 hidden neurons. The simulation is performed with two GPUs (Nvidia RTX 3090) configurations. The comparison of classification accuracy is summarized in Table 4. We examined two Ap-LBP variations, Ap-LBP $^{(1)}$ and Ap-LBP $^{(2)}$ with one and two approximated bits, respectively. Based on the obtained results, the Ap-LBP shows a minor accuracy degradation compared to counterpart networks while providing significant energy-product-delay reduction as discussed earlier. As can be seen Ap-LBP $^{(2)}$ achieves an accuracy of 90.3% on the SVHN dataset, while the LBCNN and LBPNet show 94.50% and 92.90% accuracy, respectively. In addition, Ap-LBP $^{(1)}$ and Ap-LBP $^{(2)}$ achieve an accuracy of 75.3% and 73.2% on the CIFAR-10 dataset, while the LBCNN and LBPNet show 92.99% and 74.1% accuracy, respectively. For the largest under-test dataset, i.e., CIFAR-100, Ap-LBP $^{(1)}$ and Ap-LBP $^{(2)}$ achieve respectively 60.41% and 58.5% accuracy, while the baseline accuracy is 69.55%. We acknowledge the accuracy degradation of Ap-LBP compared to other discussed models in some data-sets and the fact that it can vary across applications. Instances, where this might be acceptable, include predictive maintenance (e.g., predict failures), environmental monitoring (e.g., weather monitoring), smart home systems (e.g., smart thermostats or lighting systems), and agricultural AI applications (e.g., crop health monitoring systems). In these cases, the benefits, such as reduced latency, and cost savings as discussed in Section 3 can offset lower accuracy.

Table 4: Inference accuracy (%) of LBP networks vs. CNN.

Model	MNSIT	FashionMNIST	SVHN	CIFAR-10	CIFAR-100
Baseline [43]	99.48	94.44	95.21	92.95	69.55
BNN [44]	98.60	91.86	97.49	89.85	-
BinaryConnect [45]	98.99	-	97.85	91.73	62.3
LBCNN [15]	99.51	-	94.50	92.99	71.12
LBPNet [36]	99.50	90.61	92.90	74.1	-
Ap-LBP$^{(1)}$	99.21	89.99	91.67	75.3	60.41
Ap-LBP$^{(2)}$	97.98	86.93	90.3	73.2	58.5

7 Conclusion

This paper first presented an approximate and multiply–accumulate-free deep neural network model named Ap-LBP for efficient feature extraction. We then developed a comparator-based near-sensor processing local binary pattern accelerator (NS-LBP) and a parallel in-memory LBP algorithm to process images near the sensor based on the Ap-LBP. The results on MNIST and SVHN datasets demonstrate minor accuracy degradation compared to baseline CNN and LBP-network models, while NS-LBP achieves 1.25-GHz and an energy-efficiency of 37.4 TOPS/W. NS-LBP reduces energy consumption and execution time by a factor of 2.2× and 4× compared to a recent LBP-based network.

References

[1] T.-H. Hsu, Y.-C. Chiu, W.-C. Wei, Y.-C. Lo, C.-C. Lo, R.-S. Liu, K.-T. Tang, M.-F. Chang, and C.-C. Hsieh, "Ai edge devices using computing-in-memory and processing-in-sensor: from system to device," in *2019 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2019, pp. 22–5.

[2] T. Yamazaki, H. Katayama, S. Uehara, A. Nose, M. Kobayashi, S. Shida, M. Odahara, K. Takamiya, Y. Hisamatsu, S. Matsumoto, L. Miyashita, Y. Watanabe, T. Izawa, Y. Muramatsu, and M. Ishikawa, "4.9 a 1ms high-speed vision chip with 3d-stacked 140gops column-parallel pes for spatio-temporal image processing," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 82–83.

[3] T.-H. Hsu, Y.-R. Chen, R.-S. Liu, C.-C. Lo, K.-T. Tang, M.-F. Chang, and C.-C. Hsieh, "A 0.5-v real-time computational cmos image sensor with programmable kernel for feature extraction," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 5, pp. 1588–1596, 2020.

[4] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2017, pp. 288–301.

[5] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *2018 ACM/IEEE 45Th annual international symposium on computer architecture (ISCA)*. IEEE, 2018, pp. 383–396.

[6] S. Angizi, S. Tabrizchi, and A. Roohi, "Pisa: A binary-weight processing-in-sensor accelerator for edge image processing," *arXiv preprint arXiv:2202.09035*, 2022.

[7] S. Sheikhfaal, S. Angizi, and R. F. DeMara, "Energy-efficient recurrent neural network with mram-based probabilistic activation functions," *IEEE Transactions on Emerging Topics in Computing*, 2022.

[8] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 481–492.

[9] Z. Jiang, S. Yin, J.-S. Seo, and M. Seok, "C3sram: An in-memory-computing sram macro based on robust capacitive coupling computing mechanism," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 7, pp. 1888–1897, 2020.

[10] S. Yin, Z. Jiang, J.-S. Seo, and M. Seok, "Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 6, pp. 1733–1743, 2020.

[11] K. Lee, J. Jeong, S. Cheon, W. Choi, and J. Park, "Bit parallel 6t sram in-memory computing with reconfigurable bit-precision," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[12] F. Juefei-Xu, V. Naresh Boddeti, and M. Savvides, "Local binary convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 19–28.

[13] S. S. Sarwar, P. Panda, and K. Roy, "Gabor filter assisted energy efficient fast learning convolutional neural networks," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2017, pp. 1–6.

[14] J.-H. Lin, J. Lazarow, A. Yang, D. Hong, R. Gupta, and Z. Tu, "Local binary pattern networks," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, pp. 825–834.

[15] F. Juefei-Xu, V. Naresh Boddeti, and M. Savvides, "Local binary convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 19–28.

[16] S. Angizi, Z. He, A. S. Rakin, and D. Fan, "Cmp-pim: an energy-efficient comparator-based processing-in-memory neural network accelerator," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[17] K. Bong, S. Choi, C. Kim, D. Han, and H.-J. Yoo, "A low-power convolutional neural network face recognition processor and a cis integrated with always-on face detector," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 1, pp. 115–123, 2017.

[18] P. Bhowmik, M. J. H. Pantho, and C. Bobda, "Visual cortex inspired pixel-level re-configurable processors for smart image sensors," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–2.

[19] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy, "X-sram: Enabling in-memory boolean computations in cmos static random access memories," *IEEE TCASI*, vol. 65, pp. 4219–4232, 2018.

[20] H. Xu, Z. Li, N. Lin, Q. Wei, F. Qiao, X. Yin, and H. Yang, "Macsen: A processing-in-sensor architecture integrating mac operations into image sensor for ultra-low-power bnn-based intelligent visual perception," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 2, pp. 627–631, 2020.

[21] S. Park, J. Cho, K. Lee, and E. Yoon, "7.2 243.3 pj/pixel bio-inspired time-stamp-based 2d optic flow sensor for artificial compound eyes," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 126–127.

[22] H. Xu, N. Lin, L. Luo, Q. Wei, R. Wang, C. Zhuo, X. Yin, F. Qiao, and H. Yang, "Senputing: An ultra-low-power always-on vision perception chip featuring the deep fusion of sensing and computing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 1, pp. 232–243, 2021.

[23] Z. Li, H. Xu, L. Luo, Q. Wei, and F. Qiao, "A 5.9 μw ultra-low-power dual-resolution cis chip of sensing-with-computing for always-on intelligent visual devices," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[24] H. Xu, M. Nazhamaiti, Y. Liu, F. Qiao, Q. Wei, X. Liu, and H. Yang, "Utilizing direct photocurrent computation and 2d kernel scheduling to improve in-sensor-processing efficiency," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[25] H. Xu, Z. Liu, Z. Li, E. Ren, M. Nazhamati, F. Qiao, L. Luo, Q. Wei, X. Liu, and H. Yang, "A 4.57 μw@ 120fps vision system of sensing with computing for bnn-based perception applications," in *2021 IEEE Asian Solid-State Circuits Conference (A-SSCC)*. IEEE, 2021, pp. 1–3.

[26] M. Abedin, A. Roohi, M. Liehr, N. Cady, and S. Angizi, "Mr-pipa: An integrated multi-level rram (hfox) based processing-in-pixel accelerator," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, pp. 1–1, 2022.

[27] M. Chu, B. Kim, S. Park, H. Hwang, M. Jeon, B. H. Lee, and B.-G. Lee, "Neuromorphic hardware system for visual pattern recognition with memristor array and cmos neuron," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 4, pp. 2410–2419, 2014.

[28] K. Bong, S. Choi, C. Kim, S. Kang, Y. Kim, and H.-J. Yoo, "14.6 a 0.62 mw ultra-low-power convolutional-neural-network face-recognition processor and a cis integrated with always-on haar-like face detector," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 248–249.

[29] S. Yin, Z. Jiang, M. Kim, T. Gupta, M. Seok, and J.-S. Seo, "Vesti: Energy-efficient in-memory computing accelerator for deep neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 48–61, 2019.

[30] J. Wang, X. Wang, C. Eckert, A. Subramaniyan, R. Das, D. Blaauw, and D. Sylvester, "A 28-nm compute sram with bit-serial logic/arithmetic operations for programmable in-memory vector computing," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 76–86, 2019.

[31] W. Simon, J. Galicia, A. Levisse, M. Zapater, and D. Atienza, "A fast, reliable and wide-voltage-range in-memory computing architecture," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[32] J. Yang, Y. Kong, Z. Wang, Y. Liu, B. Wang, S. Yin, and L. Shi, "24.4 sandwich-ram: An energy-efficient in-memory bwn architecture with pulse-width modulation," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2019, pp. 394–396.

[33] W.-T. Kim, H. Lee, J.-G. Kim, and B.-G. Lee, "An on-chip binary-weight convolution cmos image sensor for neural networks," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 8, pp. 7567–7576, 2020.

[34] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, "Redeye: analog convnet image sensor architecture for continuous mobile vision," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 255–266, 2016.

[35] D. Wang, C.-T. Lin, G. K. Chen, P. Knag, R. K. Krishnamurthy, and M. Seok, "Dimc: 2219tops/w 2569f2/b digital in-memory computing macro in 28nm based on approximate arithmetic hardware," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 266–268.

[36] J.-H. Lin, J. Lazarow, A. Yang, D. Hong, R. Gupta, and Z. Tu, "Local binary pattern networks," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, pp. 825–834.

[37] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[38] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[39] S. Xu, X. Chen, Y. Wang, Y. Han, X. Qian, and X. Li, "Pimsim: A flexible and detailed processing-in-memory simulator," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 6–9, 2018.

[40] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "Cacti 5.1," Technical Report HPL-2008-20, HP Labs, Tech. Rep., 2008.

[41] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A mixed-signal binarized convolutional-neural-network accelerator integrating dense weight storage and multiplication for reduced data movement," in *2018 IEEE Symposium on VLSI Circuits*. IEEE, 2018, pp. 141–142.

[42] Synopsys, Inc., "Synopsys design compiler, product version 14.9.2014," 2014.

[43] P. Gavrikov and J. Keuper, "Cnn filter db: An empirical investigation of trained convolutional filters," *arXiv preprint arXiv:2203.15331*, 2022.

[44] P. Sermanet, S. Chintala, and Y. LeCun, "Convolutional neural networks applied to house numbers digit classification," in *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*. IEEE, 2012, pp. 3288–3291.

[45] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *arXiv preprint arXiv:1511.00363*, 2015.

Shaahin Angizi (SM'22) is currently an Assistant Professor in the Department of Electrical and Computer Engineering, New Jersey Institute of Technology (NJIT), Newark, NJ, USA, and the director of the Advanced Circuit-to-Architecture Design Laboratory. He completed his doctoral studies in Electrical Engineering at the School of Electrical, Computer and Energy Engineering, Arizona State University (ASU), Tempe, AZ in 2021. His primary research interests include ultra-low-power in-memory computing based on volatile & non-volatile memories, in-sensor computing for IoT, brain-inspired (neuromorphic) computing, and accelerator design for deep neural networks and bioinformatics. He has authored and co-authored +90 research papers in top-ranked journals and EDA conferences. He is the recipient of the Best Ph.D. Research Award (1st-place) of Ph.D. Forum at IEEE/ACM DAC in 2018, two Best Paper Awards of IEEE ISVLSI in 2017 and 2018, and Best Paper Award of ACM GLSVLSI in 2019.

Mehrdad Morsali is currently a Ph.D. student in the Department of Electrical and Computer Engineering, New Jersey Institute of Technology (NJIT), Newark, NJ, USA. He completed his masters' degree in electrical engineering at Shahid Beheshti University (SBU), Tehran, Iran, in 2020. He received his bachelors' degree in electrical engineering from the Urmia University, Urmia, Iran, in 2014. His current research interests include in-sensor Computing and Non-volatile Memories.

Sepehr Tabrizchi (S'21) is currently a Ph.D. student at the School of Computing, University of Nebraska-Lincoln, USA. He received a masters' degree in computer systems architecture from Azad University, Science and Research Branch, Tehran, Iran, in 2018 and a Bachelor of Engineering degree in computer engineering from the Islamic Azad University Najafabad Branch, Iran. His research interests include ternary logic, VLSI design, non-volatile memories, and brain-inspired (neuromorphic) computing.

Arman Roohi (SM'23) is currently an assistant professor with the School Computing, University of Nebraska-Lincoln, USA. Before joining UNL in 2020, he was a postdoctoral research fellow at the University of Texas at Austin. He received Ph.D. degree in Computer Engineering at the University of Central Florida, Orlando, FL, USA, in 2019. His research interests span the areas of design of cross-layer co-design for implementing complex machine learning tasks secure computation, including hardware security, and the security of artificial intelligence, reconfigurable and adaptive computer architectures, and beyond CMOS computing. He has completed over 50 publications on these topics, including best paper recognition, book chapters, and STEM curricular development. He received Ph.D. Forum at DAC 2018 Scholarship, Frank Hubbard Engineering Endowed Scholarship in 2018, best paper recognitions in IEEE Transactions on Emerging Topics in Computing in 2019, and paper of the month at IEEE Transactions on Computers in 2017.