PiPSim: A Behavior-Level Modeling Tool for CNN Processing-in-Pixel Accelerators

Arman Roohi, *Senior Member, IEEE*, Sepehr Tabrizchi, *Student Member, IEEE*, Mehrdad Morsali, *Student Member, IEEE*, David Z. Pan, *Fellow, IEEE*, and Shaahin Angizi, *Senior Member, IEEE*

***Abstract*—Convolutional neural networks (CNNs) have been gaining popularity in recent years, and researchers have designed specialized architectures to speed up the inference process. However, despite the promising potential of processing near-/in- sensor architectures actively explored in the visual Internet of Things, there is still a need to develop a behavior-level simulator to model performance and facilitate early design exploration. This article proposes a stand-alone simulation platform for processing-in-pixel (PiP) systems, namely, PiPSim. It offers a flexible interface and a wide range of design options for customizing the efficiency and accuracy of PiP-based accelerators using a hierarchical structure. Its organization spans from the device level, e.g., memory technology, upward to the circuit level, e.g., compute-add on architecture, and then to the algorithm level, e.g., DNN workloads. PiPSim realizes instruction-accurate evaluation of circuit-level performance metrics as well as learning accuracy at run-time. Compared to SPICE simulation, PiPSim achieves over 25 000× speed-up with less than a 2.5% error rate on average. Furthermore, PiPSim can optimize the design and estimate the tradeoff relationships among different performance metrics.**

***Index Terms*—Design optimization, energy efficiency, neural network, numerical simulation, processing-in-sensor (PiS).**

I. Introduction

CURRENTLY, nearly 90% of the data generated by the Internet of Things (IoT) is not analyzed or processed, mainly due to the insufficient computing ability of state-of-the-art processors/memory of small area and power-restricted IoT devices and memory/compute-intensive convolutional neural network (CNN) algorithms [1], [2]. In order to resolve these issues, computing architectures must shift from a cloud-centric approach to a thing-centric (data-centric) approach, where the IoT node processes the sensed data. processing-in-pixel (PiP) paradigm has been recently set forth as a new class in rapidly developing near-/in-sensor and in-memory computing platforms to process pre-analog-to-digital converter (pre-ADC) data before converting and transmitting it to the on/off-chip processor [3], [4], [5], [6]. Due to PiP's limited resources, it is inefficient to deploy all neural network layers into the pixel array. Thus, almost all studies accelerated the first layers in an analog domain and submitted the rest to the digital neural network accelerator. Block diagrams of different visual architectures are shown in Fig. 1. The PiP paradigm 1) significantly reduces the power consumption of converting photo-currents into pixel values used for image processing; 2) accelerates data processing allowing simultaneous sensing and computing; and 3) alleviates the memory bottleneck problem [7], [8]. RedEye [9] implemented convolution operation using charge-sharing tunable capacitors. Compared to a CPU/GPU, this design sacrifices accuracy in favor of energy savings. Nevertheless, the energy required per frame increases dramatically by 100× for high-accuracy computation. MACSen [10], as a PiP platform, processes the 1st-convolutional layer of binary-weight neural networks (BWNNs) using the correlated double sampling method at a speed of 1000 fps. However, it suffers from a considerable area overhead and high power consumption due to the SRAM-based PiP. In [4], a PiP-based architecture has been proposed that operates convolution in an analog domain to reduce power consumption. Nevertheless, this design requires four photodiodes and four capacitors per pixel, resulting in an overall area and power overhead.

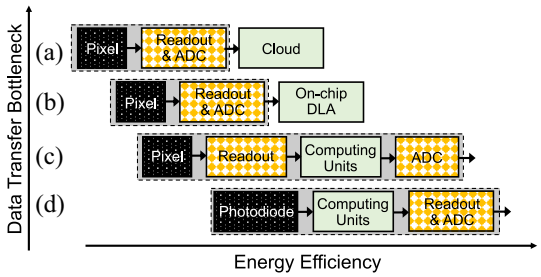

Fig. 1. Different visual systems: (a) conventional architecture, (b) PnS architecture [8], [11], [12], [13], [14], alongside DLA, (c) PiS architecture [10], [15], [16], [17], and (d) PiP architecture [4], [18], [19], [20], where black and dotted orange boxes indicate the pixel and the sensors, respectively, and green boxes represent where the computing is performed.

Manuscript received 10 April 2023; revised 24 June 2023; accepted 2 August 2023. Date of publication 15 August 2023; date of current version 26 December 2023. This work was supported in part by the National Science Foundation under Grant 2216772 and Grant 2216773. This article was recommended by Associate Editor C. Bolchini. *(Corresponding author: Arman Roohi.)*

Arman Roohi and Sepehr Tabrizchi are with the School of Computing, University of Nebraska–Lincoln, Lincoln, NE 68588 USA (e-mail: aroohi@unl.edu).

Mehrdad Morsali and Shaahin Angizi are with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07103 USA (e-mail: shaahin.angizi@njit.edu).

David Z. Pan is with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712 USA (e-mail: dpan@mail.utexas.edu).

Digital Object Identifier 10.1109/TCAD.2023.3305574

As various image processing applications with distinct workload sizes and memory access patterns are expected to benefit from PiP at the edge devices, selecting the right design for a particular application is vital but challenging. Moreover, it is imperative to establish uniform evaluation conditions to make an impartial choice between available design options. To enable design space exploration and the cross-technology comparison of various PiP designs, we have developed a bottom-up behavior-level modeling tool for PiP accelerators named **PiPSim**. The extensible PiPSim enables IC designers to develop a configurable PiP design to emulate *components*, such as sensing units, shared data bus, etc., and *operations* like parallel in-sensor multiply–accumulate (MAC) to evaluate various performance metrics and accuracy. This article makes the following contributions.

1) Assessment of the sensing/processing time, power consumption, and throughput of PiP chips considering various pixel structures and specific design features before moving toward the fabrication stage.
2) Design space exploration of PiP platforms to attain an optimized device/circuit/architecture combination to improve the performance parameters.
3) Consolidation of circuit-architecture methods realizing a fully stand-alone and automatic simulator, not relying on multiple separate simulators.

The remainder of this article is organized as follows. In Section II, a brief review of different simulators, especially processing near/in memory simulators, is provided. Section III introduces our PiPSim framework, highlighting its key features and design principles. The simulation results and the detailed analysis are summarized in Section IV, where we argue that the PiPSim Framework represents an important step forward in the development of computational tools for simulating complex processing in sensors, especially for CNN networks. Finally, Section V concludes this article by emphasizing the advantages and features of the proposed architecture.

II. Background

Although there is no prior behavior-/system-level performance and energy evaluation framework for PiP architectures, researchers have developed a myriad of in-house memory evaluation and processing-in-memory (PIM) tools, including circuit simulators, RTL-based and behavioral simulation frameworks. These efforts can be categorized into three groups.

1) *Memory evaluation tools* such as HP-Lab's Cacti [21], which was originally designed to evaluate the DRAM or SRAM-based memory and cache. Nonvolatile memory simulator (NVSIM) [22] is a circuit-level simulator that has been developed on top of Cacti to perform performance, energy, and area estimations for nonvolatile memories (NVMs), such as STT-RAM, PCRAM, ReRAM, and legacy NAND Flash. It facilitates design space exploration for PIM architectures and helps optimize memory systems by analyzing tradeoffs between energy, latency, and area. Nonvolatile main memory (NVmain) [23] models the main memory based on DRAM and emerging NVM NPSabbrpl, such as PCRAM, STT-RAM, and hybrid designs. NVmain can be integrated with various microarchitecture simulators, such as gem5, Sniper, and McSimA+. It provides a detailed and flexible framework for simulating the timing, power, and energy characteristics of NVM technologies, making it an important tool for researchers to study and understand the potential benefits and drawbacks of using these memory systems. Ramulator [24] was proposed as an open-source, cycle-level, and extensible DRAM simulator to evaluate different currently used DRAM standards as well as those of the future. It has become a widely used tool for modeling and evaluating the performance of a wide range of DRAM technologies, including DDRx, LPDDRx, GDDRx, and HBMx.
2) *Analog PIM accelerator simulators* such as MNSIM (memristor-based neuromorphic computing system simulator) [25] as a fast behavior level simulation platform to estimate and optimize the performance of a memristor-based neuromorphic accelerator. It enables designers to evaluate the performance, energy consumption, and area of neuromorphic systems, such as memristive crossbar arrays, to develop more efficient PIM architectures for artificial intelligence (AI) and machine learning applications. As a circuit-level macro model, NeuroSim [26] has been developed to be integrated with the learning algorithm also supporting a wide variety of emerging memory technologies. IBM Analog Hardware Acceleration Kit (AIHWKIT) [27] is an open-source analog AI simulation toolkit with a convenient PYTORCH interface presented to simulate analog crossbar arrays.
3) *Digital PIM accelerator simulators*, such as evaluating cache-in-memory (Eva-CiM) [28] that leverages several existing memory and micro-architecture modeling tools, with GEM5 as the backbone [29], to reliably predict performance and energy consumption of digital PIM modules. PIM simulator (PIMSim) [30], simulator-to-processing in memory (Sim2PIM) [31], and computing-in-memory simulator (CIM-SIM) [32] have been designed as a full-system and configurable PIM simulator to ease research from abstract to system-level. SIM2PIM translates general-purpose processor simulation models into PIM models. It enables the evaluation of PIM architectures for a wide range of applications and helps optimize the performance and energy efficiency of these systems. In the CIM-SIM framework, processing elements are integrated within memory cells or arrays, which supports various emerging memory technologies and provides insights into the performance, energy consumption, and area tradeoffs for different CIM designs.

III. PiPSim Framework

The **PiPSim** framework takes users' needs and inputs, including target energy, hardware constraints, etc., and

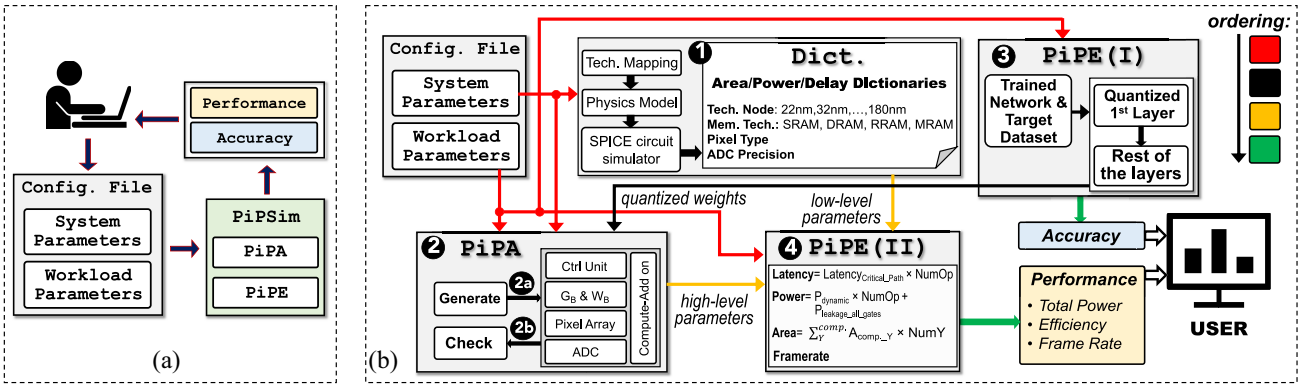

Fig. 2. (a) Overview of the proposed PiPSim. (b) High-level and circuit block diagram of a PiP architecture w.r.t. the input configuration file.

TABLE I
CONFIGURATION INPUTS OF PIPSIM

	Inputs	Default Value	Allowed Value
Workload Parameters	Filter Size ($R \times S$)	3×3	$\{R, S \in \mathbb{N}\}$
	Number of Filters	32	$\in \{1, 2, \ldots, 128\}$
	Sliding Window	1	$\in \{1, 2, \ldots, r\}$
	Channel Pruning⋆	0	$\in \{0, 1, 2\}$
	Quantization Level	Binary	$\in$\{Binary, Ternary, Quinary\}
System Parameters	Memory Technology	SRAM	$\in$\{SRAM,DRAM,ReRAM, MRAM\}
	Pixel Array Size	64×64	$\in \{8 \times 8, \ldots, 600 \times 600\}$
	Pixel Type	3T	$\in$\{3T, 4T, customized one\}
	ADC Precision	8-bit	$\in \{1, 2, \ldots, 8\}$
	Box Size	3×3	$\in 2\mathbb{N} + 1$
	Parallelism Level*	0	$\in \{0, \ldots, K\}$

⋆ Using colorful inputs with three R, G, and B channels, Channel Pruning is set to 0, while Grayscale images leverage Channel Pruning= 2.
* Parallelism Level is determined by applying a number of different filters simultaneously, which increases efficiency at the cost of hardware overhead.

generates a close approximation of performance metrics, as depicted in Fig. 2(a). PiPSim is composed of three primary components: 1) *prestored dictionaries* (Dicts.); 2) processing-in-pixel architecture (PiPA) module; and 3) processing-in-pixel examiner (PiPE), consisting of two procedures, PiPE(I) and PiPE(II) to assess accuracy and energy/delay parameters, as shown in Fig. 2(b). First, device/circuit features are analyzed offline and stored in the Dicts. ① This step can be coordinated with Cacti and NVSIM. The PiPA module takes both workloads and system parameters ② listed in Table I, to form a PiP structure. The PiPE (I) ③ component takes the workload parameters and quantizes the first layer's weights of the neural network. Then the quantized weights are fed to PiPA ②. Meanwhile, PiPE(I) modifies the neural network with the new first layer and performs the accuracy test. Herein, PiP architectures only target the first layer, and the remaining layers will be calculated by analog/digital deep-learning accelerators. Within PiPA, the Generate procedure (2a) determines buffer sizes and the number of compute add-ons (CAs) based on the applied parameters. If any input is missing, the default value is leveraged. After having all the required components, e.g., pixel array size, etc., the Generate module creates the proper Control (Ctrl.) Unit, including row and column decoders, places all the components together. To minimize the error possibility or/and design violation, the generated architecture's functionality is validated by the Check procedure (2b) through several if/else conditions, detailed in Algorithm 1. Since this step is an iterative process, it continues till the desired error-free design is achieved. It is worth mentioning that the achieved PiP design is not the best-optimized design, and additional optimization steps could be considered by the users. If all the design criteria are satisfied, the generated high-level parameters are taken by PiPE(II) ④. Moreover, it takes the user's and low-level (LL) parameters and evaluates the overall system's performance, including latency, power consumption, and framerate.

A. Hierarchical Bottom-Up Methodology

PiPSim includes several predefined libraries and components to support different possible PiP architectures. However, users can also integrate the power, latency, and other parameters of customized components into PiPSim. The proposed PiPSim is versatile and can be readily exposed to the well-developed Cacti [21] and NVSim [22] C++ libraries to add new components.

1) Device Model: In order to determine process parameters, **PiPSim** uses device data from the international technology roadmap for semiconductor (ITRS) report. It supports three types of transistors, including high performance, low power, and low standby power, which cover process nodes $\in$ {180, 90, 65, 45, 32, 22} nanometers.

2) Memory Technology: The device-level model of the supported memory technologies is developed/extracted from various models and assessments. For spin-transfer torque magnetic RAM (STT-MRAM) and spin-orbit torque magnetic RAM (SOT-MRAM) as shown in Fig. 3(a) and (b), we jointly use the nonequilibrium green's function (NEGF) and Landau–Lifshitz–Gilbert (LLG) equations to model the bit-cell, developed in preliminary works [33]. We leverage the 1T1R ReRAM model developed in our experimental TiN/Ti/HfO_2/TiN bit-cell [18] as shown in Fig. 3(c). The default 6T SRAM cell configuration is adopted from NVSim [22]. The eDRAM cell parameters are adopted and scaled from Rambus [34]. The FEFET model is inevitability skipped in this work due to the immaturity of the experimentally benchmarked data.

3) Pixel Type: The PiPSim simulator supports two highly used pixel types, 3- and 4-transistors (3T and 4T), including one photodiode, as shown in Fig. 4(a) and (b), respectively. These pixels can generate only positive currents on bit-line (BL); therefore, we modified them to generate both negative and positive currents. The modified structure is shown

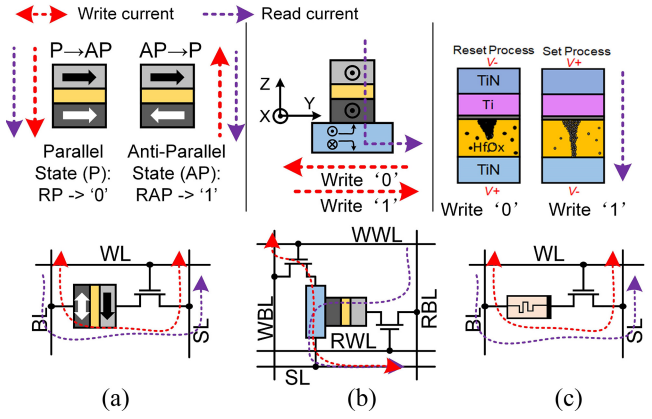

Fig. 3. (a) STT-MRAM, (b) SOT-MRAM, and (c) ReRAM models.

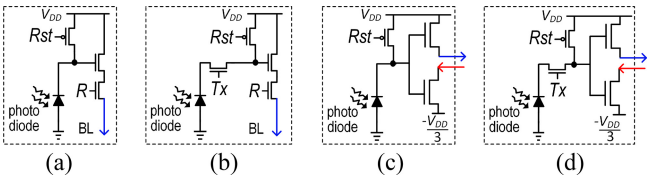

Fig. 4. Widely used pixels (a) 3T and (b) 4T pixels, and (c) and (d) are two modified pixel structures that support negative and positive currents.

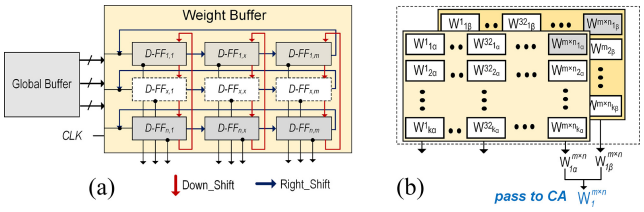

Fig. 5. (a) Global buffer and a weight buffer with down and right shift capabilities. (b) 2-bit weights to represent ternary weights using two W_Bs.

in Fig. 4(c), where the row selector transistor (R) is moved to another new building block, i.e., CA, explained in the following. It should be mentioned in standard PiP designs, there is no dedicated memory to store the pixel values. Hence, to ensure proper functionality regarding networks with a large number of filters, e.g., 32, designers limit to the 4T-pixel or similar but customized pixel designs. The reason is that in the 4T-pixel, the stored value in a capacitor remains unchanged long enough after evaluation.

4) Storage Component: PiPSim has two different reconfigurable memory banks named *Global Buffer* (G_B) and *Weight Buffer* (W_B), shown in Fig. 5(a). The G_B is responsible for storing all the weights concerning the quantization level. The size of G_B directly relates to the number of filters (K) in the first layer, the spatial dimension of filters ($R \times S$), and the weights' precision. The second memory element, W_B, stores only one filter; thus, its size is much smaller than G_B storage and is determined by the filter size, quantization, and parallelism levels. In addition, W_B is capable of shifting right and down to implement stride behavior, depicted in Fig. 5(a). The stride window can be defined by a designer or set to one as the default value. Herein, the connections between the weight buffer and pixels are hardwired, which

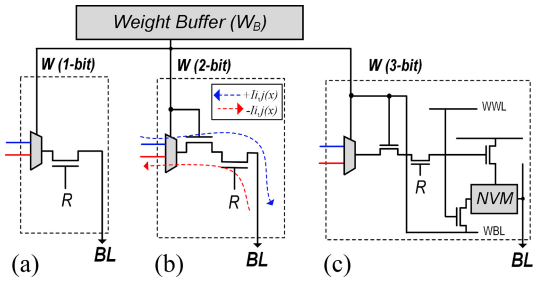

Fig. 6. Three different compute-add on (CAs) architectures regarding the (a) binary, (b) ternary, and (c) quinary quantization levels.

provides an efficient convolution-in-pixel scheme. The desired PiP accelerator reuses both input feature map (ifmap) and weights data types to form input stationery and weight stationary dataflows, which reduces the overall data transfer significantly. We need one-to-several W_B components to realize different quantization and parallelism levels. For instance, to map ternary weights, $\in \{-1, 0, 1\}$, into the memory, we need two W_Bs, each of which output functions as a control signal for transistors in the CA. The W_B storage is directly connected to the pixel array and CA, shown in Fig. 5(b).

5) Compute Add-On: The CA is utilized to support the computation, i.e., multiplication and addition, required for event detection and object classification tasks. Every CA is controlled by the pixel's voltage (V_{PD}), shown in Fig. 4, and generates positive or negative current flow on the shared BL regarding the value stored in the W_B. Although PiPSim supports three possible quantization levels, i.e., binary [Fig. 6(a)], ternary [Fig. 6(b)], and quinary [Fig. 6(c)], users can perform a design space exploration by adopting other quantization approaches and implementations. The blue line in this figure indicates the positive current, while the red line represents the negative current, both of which are controlled by a multiplexer. In both (b) and (c), there is a transistor included for the purpose of interrupting the current flow in order to produce zero current (weight) that reduces power consumption as well. Furthermore, in (c), a resistor memory (ReRAM) is utilized to produce two distinct levels of output current. To this end, PiPSim readily takes the circuit-level performance parameters such as power and delay of new modified CAs in the Dicts. block. from user.

6) Enabling Binary/Ternay/Quinary Weight Neural Network: As previously mentioned, every pixel connects to one-to-several W_Bs, consisting of different coefficient matrices, e.g., α, β, and ϕ, responsible for generating appropriate weights summarized in tables in Fig. 7. For ternary and quinary weights, the value α is specified to generate a current flow in a pixel or not. If α is zero, it disables the pixel, and other coefficients are unimportant. The current direction, i.e., negative or positive [in Fig. 7(b) and (c)], and the current magnitude [in Fig. 7(c)] are determined by β and ϕ, respectively. With respect to the weights, the power consumption, and all the functionalities are collected and held in the Dict. blocks.

In addition to validating the functionality of the pixel and CA, we analyzed the proposed pixel under various conditions.

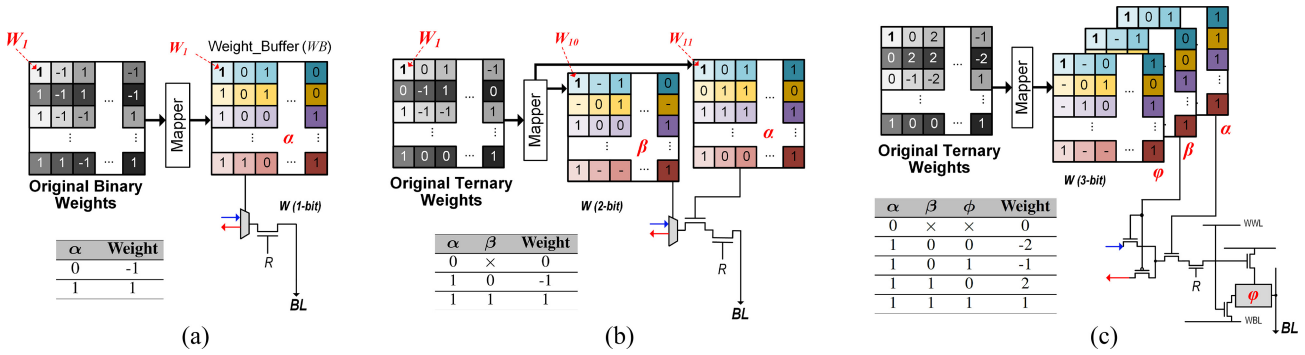

Fig. 7. Illustration of (a) binary, (b) ternary, and (c) quinary weights, regarding the stored α, β, and ϕ.

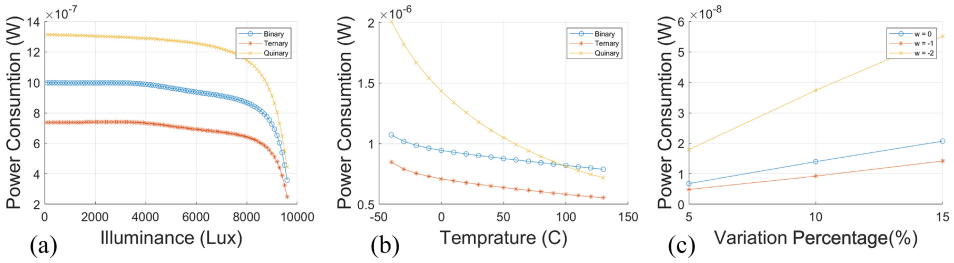

Fig. 8. Relationship between the power consumption and three metrics, including (a) illuminance, (b) temperature, and (c) mismatch.

The outcomes of this analysis are presented in Fig. 8. In Fig. 8(a), the power consumption of the pixel is compared to the illuminance. The results indicate that, in general, ternary weight consumes less power than other weight precisions. This is because, in one-third of instances, there is no current in the output. As the light intensity increases, all pixels consume less power in some cases, such as 10 000 lux. This is due to the increase in the reverse current of the photodiode, leading to the complete discharge of the pixel's capacitor during the evaluation phase. As a result, the pixel is unable to generate any current, regardless of weight values. In Fig. 8(b), the power and temperature have a direct relationship, with temperature varying from -40 °C to 140 °C. Finally, in Fig. 8(c), the correctness of the pixel operations is demonstrated by the results obtained under the presence of 15% process variation in transistor sizing over 1000 simulation runs.

7) Readout and ADC Components: To consider both efficiency and accuracy, the analog-to-digital conversion (ADC) and readout circuitry are customized and redesigned based on the user's parameters. One optimization here is that an ADC is shared and utilized for several columns, which is determined by the filter's width. For a filter size of $R \times S$, the number of required ADCs reduces to $\lceil W/S \rceil$, where W is the width of the input feature map (ifmap). Herein, all the MACs for the first layer are performed in the analog domain and the ADC precision can be directly configured according to the user's needs. Due to the fact that most CNNs can be quantized into 8-bit fixed-point and provide acceptable accuracy [35], PiPSim limits the precision of ADC to 8 bits, which can vary between 1 and 8 bit. The performance models of several popular ADC designs [36] have been integrated into PiPSim. It should be noted that to implement the correct shift, in addition to shifting W_B, the order of connections among columns should change. To do so, X switches are positioned between every two columns, where $X = W - 2$. Another optimization is achieved by changing the row-wise output feature map (ofmap) processing to column-wise, reducing the required number of switch operations.

PiPSim takes into account the activation functions using the readout peripheral circuitry. Specifically, two activation functions are considered: rectified linear unit (ReLU) and Sign functions. Depending on the designer's preference, either of these activation functions can be used. If the Sign activation function is chosen, there is no need for ADC because it works directly with digital signals. This can simplify the design process and potentially reduce the system's complexity at the cost of accuracy loss. Conversely, if the designer decides to use the ReLU activation function, some additional steps may be required for implementation. In order to achieve higher accuracy, the reference voltage of the ADC may need to be adjusted. Alternatively, a 2^{n-1} bit ADC can be used instead of a 2^n bit ADC to reduce power consumption. A simple reconfigurable ADC with sign and ReLU activation function is shown in Fig. 9. Careful consideration should be given to calculating the power consumption of such devices. For instance, if the user uses the sign function 80% of the time and the ReLU function for the remaining 20%, the reported power should adhere to the same usage.

8) Control Units: The control unit consists of several components, including a row decoder, column and memory controllers, and switches for ADCs. *Row Decoder:* This component enables each row within the pixel array. The required number of decoders increases with the increase in filter height (R), whereas the I/O ports of the decoders are reduced. *Column Controller:* In the implemented PiP architecture, each column is connected to a separate V_{DD}. The column controller can turn each column ON/OFF individually, realizing a

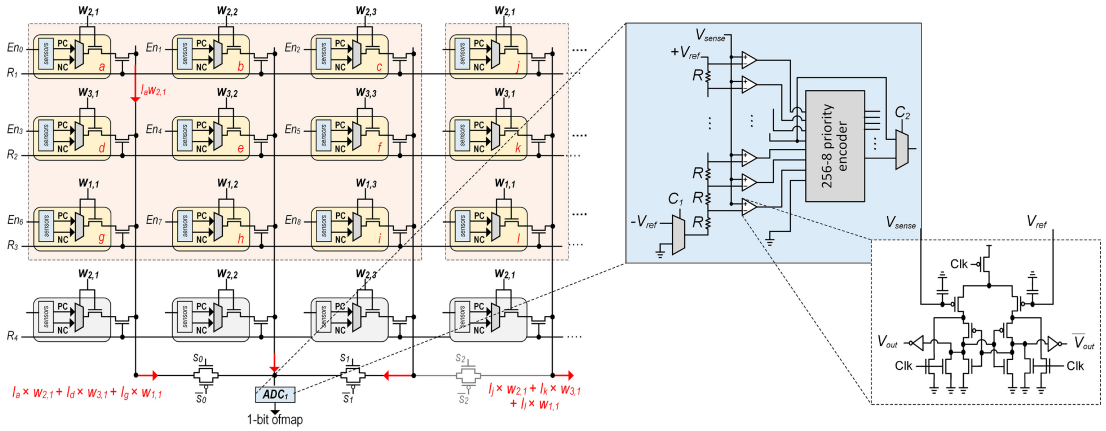

Fig. 9. Structure realizing ternary weight neural network that converts every input pixel value to a weighted current according to the stored weight that is interpreted as the multiplication; thus, convolution is readily calculated by measuring the voltage across a sensing resistor and proper signaling of the switches.

power-efficient event-detection technique. *Switches for ADCs:* With these switches, PiPSim can connect an n number of columns together to perform convolution-in-pixel by sliding weights over the input feature map (ifmap), where n is determined by the widths of the filters and the stride, as shown in Fig. 9. The switches are controlled by the *config_switch* signal. *Memory Controller:* This unit controls the shift, read, and write operations, which can vary depending on the memory technology (e.g., DRAM) and the level of parallelism. Besides, it enables row-wise parallel read/write operations corresponding to the background update step in the event-detection technique.

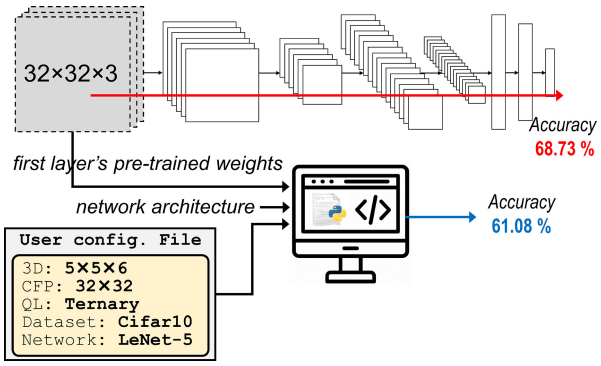

Fig. 10. PiPE(I) module takes the pretrained model and user's parameters, modifies the network, and examines the accuracy.

B. Behavior-Level Accuracy Model

To validate the correctness of modified neural networks and ensure acceptable accuracy, we developed a Python script that runs above the PyTorch library and allows users to apply their modifications (PiPE(I)). This script can be run parallel with PiPA, as shown in Fig. 2(b). The script takes three inputs: 1) a pretrained model; 2) a target dataset; and 3) user-specified parameters. The key parameters affecting accuracy are the quantization level, filter size, channel pruning, pixel array size, and stride. Since the PiPSim framework focuses on the first layer, this script can operate in parallel, and it only adjusts the first layer using the defined inputs, while the rest layers remain unchanged. Regarding filters of the first layer, the quantization level variable forces the script to quantize the weights properly. Moreover, the script makes necessary adjustments according to the ifmap. For example, if the user wants to use *Channel Pruning*= 2, the script converts a colorful input image with three channels (red, green, and blue) to grayscale (one channel) based on the unpruned channel; or/and if the input size is incompatible with the defined pixel array size, the image should be reshaped to the same height and width. For example, as depicted in Fig. 10, targeting the CIFAR-10 dataset and simple/shallow LeNet network, PiPE(I) mapped one 32×32 input to the pixel array and uploaded all six filters with the size of 5×5. This module evaluates the desired workloads by modifying LeNet's first layer based on the obtained PiP architecture and user inputs. While the original full-precision network could not provide good accuracy due to its simple architecture, after pruning channels and applying quantization, our implementation demonstrated a small accuracy loss occurred ($\sim 7\%$).

C. Putting All the Components Together

After integrating all components to meet the user's demands, the generated PiP structure resembles the schematic shown in Fig. 11, consisting of an $n \times m$, e.g., 32×32, compute focal plane array, row and column controllers, command decoder, sensor timing Ctrl., global and weight buffers, and switch/ADC circuitry. The CFP is designed to co-integrate sensing and processing for low-power but high classification accuracy applications. The output, i.e., 1st-layer of DNN, is transmitted to an on-chip deep learning accelerator (DLA) to accelerate further. Algorithm 1 illustrates the primary steps used by PiPSim, where in addition to ensuring the design correctness, some optimization steps are considered to make it more efficient.

1) Event-Detection Procedure: Since always-on edge devices usually have limited resources, this mode is developed to save power, specified by Box_Size and precision.[1] Here, all pixels are grouped by the size of *box_size*, and in each box, only the central pixel is ON. It means by choosing a

[1]This is different from the ADC precision parameter.

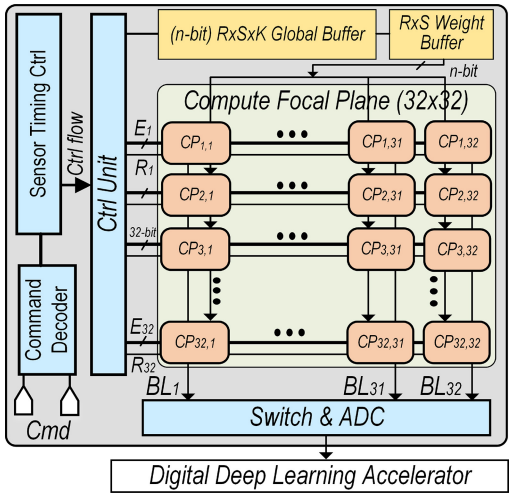

Fig. 11. Possible produced PiP architecture after determining each component by PiPA and PiPE(I) procedures.

larger *box_size*, and more power saving is achieved. Another effective parameter is *precision*, which indicates how many bits of the central pixel should compare with the previous value of the pixel. PiPSim uses the first frame as a static image for the background and performs the background subtraction process to detect moving objects. There are two more important parameters, namely, $threshold_{pixels}$ and $time_{\tau}$, that are related to the maximum number of differences in each row and the number of continuous switching to *Pixels_Enable* mode, respectively. Herein, every row containing central pixels becomes active (line 5), and all the central pixels' values compare with their previous ones in parallel. After comparison, the number of unequal pixels is stored in *num_changes* (line 7). If this value is higher than the defined $threshold_{pixels}$, this row is marked to turn ON in the `pixels enable` mode. Whenever the system switches to this mode, the value of the timer will be increased by one, which helps the system to detect an added object in the background and updates the previous values of central pixels with the new ones (line 3).

2) Pixels-Enable Procedure: This procedure takes the marked rows from the previous step. Its other input is the *power_mode* parameter with a range of [1, *box_size*]. If the *power_mode* is set to 1, only the specific pixels within the turn_on_list are enabled, while by increasing the *power_mode* value, more adjacent pixels are turned ON (lines 23 and 24). By decreasing the *power_mode* value, power consumption reduces at the cost of accuracy degradation.

3) Convolution-in-Pixel: The obtained PiP design using PiPA performs 1st-layer's convolution operations in the analog domain instantly after capturing an image that increases MACs throughput and decreases the ADC overheads. It implements the input stationary dataflow to minimize the reuse distance of ifmaps, maximizing the convolutional and ifmap data reuse. All capacitors within the $n \times n$ pixel array are written regarding the light intensity of a target image. The stored values remain almost constant for a specific time interval, which is determined by the capacitance of the capacitors. Moreover, the W_B supports the weight stationary dataflow for specific

Algorithm 1 Three Primary Procedures Deployed in PiPA and Used by PiPE(II) Modules

1: **procedure** EVENT-DETECTION
2: **if** time $\geqslant$ $time_{\tau}$: ▷ Merge steady objects with the background.
3: **update** (background)
4: **for** $i = \lfloor \frac{box\_size}{2} \rfloor + 1$ to H with step= *box_size*
5: **activate** (row_i)
6: pixel_values ← **parallel_read** ($column_{i,j}$) ▷ $j \in \{\lfloor \frac{box\_size}{2} \rfloor + 1, \ldots, W\}$, with step= *box_size*
7: num_changes ← **parallel_comp** (*precision*, pixel_values, old_values)
8: **if** num_changes $\geqslant$ $threshold_{pixels}$:
9: turn_on_list.**push** (i) ▷ i is row index.
10: **if** (length (turn_on_list) !=0)
11: time += 1 ▷ Use it to update the background.
12: **enable** PIXELS_ENABLE
13: **else**:
14: time = 0
15: **end procedure**
16: ---
17: **procedure** PIXELS_ENABLE
18: **if** (*power_mode* == *box_size*)
19: **Enable_pixel** (All)
20: **else**
21: **while** (length (turn_on_list) !=0)
22: row = turn_on_list.**pop**
23: margin = $\lfloor \frac{box\_size}{2} \rfloor$ + *power_mode* − 1
24: **Enable_pixel** (*row* − *margin* **to** *row* + *margin*)
25: **enable** CIP
26: **end procedure**
27: ---
28: **procedure** CONVOLUTION-IN-PIXEL (**CIP**)
29: **parallel_for** $k \leftarrow 1$ to K ▷ number of parallelism
30: **for** $s \leftarrow 0$ to S step= *stride* ▷ left to right
31: **config_switch** (s)
32: **for** $r \leftarrow 0$ to R step= *stride* ▷ top to bottom
33: **for** $h \leftarrow \lfloor R/2 \rfloor + 1 + r$ to $(H - \lfloor R/2 \rfloor)$ with step= R
34: **Active_Row** ($h - \lfloor R/2 \rfloor$ to $h - \lfloor R/2 \rfloor$)
35: **Calculate_CONV** ()
36: **Shift_Down** ($W_B\downarrow$, *stride*)
37: **Shift_Right** ($W_B\rightarrow$, *stride*)
38: **Load_New_Weight** ($G_B \Rightarrow W_B$)
39: **end procedure**

timeframes. The loop with variable K, which denotes the number of filters, is placed in the outermost loop (line 29). It activates R rows (line 34) and performs convolutions for all selected columns. Then shifts the same filter down (line 36) and continues the previous steps. After considering all possible movements based on the stride window, weights are shifted to the right (line 37), and so on. Since the connections between W_B's blocks and pixels are hardwired, different weights of a $R \times S$ filter are unicast to a group of pixels, while the same filter is broadcast to other groups of pixels in different columns. The spatial dimensions of a filter are represented by R and S, height and width, respectively. Thus, PiPSim can compute $R \times S \times n$ MAC operations in only one clock cycle, where $n = \lfloor W/S \rfloor$ and W is the input's width. However, applying the same filter to other R input rows needs an extra cycle. Therefore, in the worst-case scenario and to maximize weight data reuse, H cycles are required before shifting the filter's values, where H is the input's height.

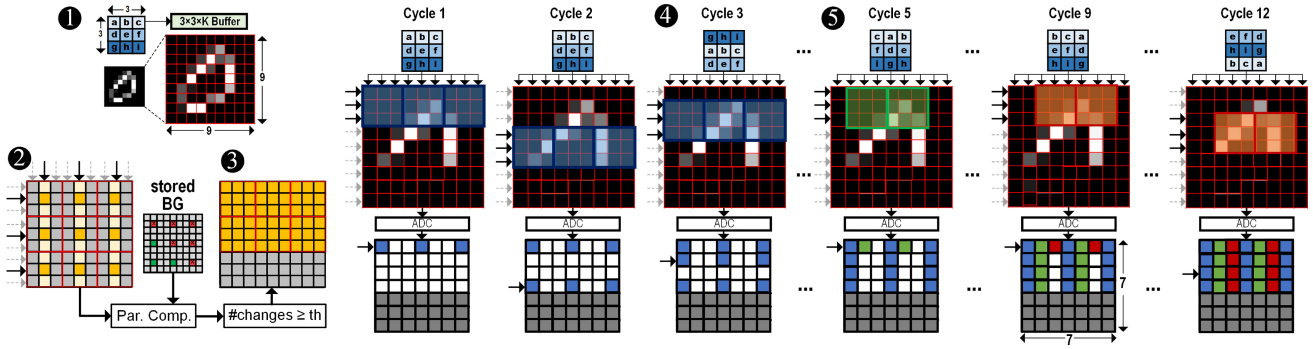

Fig. 12. Example of the proposed event-detection, and convolution-in-pixel methods with 3 × 3 filter size and 9 × 9 pixel array as inputs. Due to the lower number of changes in the last three rows than the threshold, these part is disabled, which reduces the number of convolutions and consequently, data communication. Twelve cycles are required to compute 28 MAC operations and generate the 7 × 7 of map.

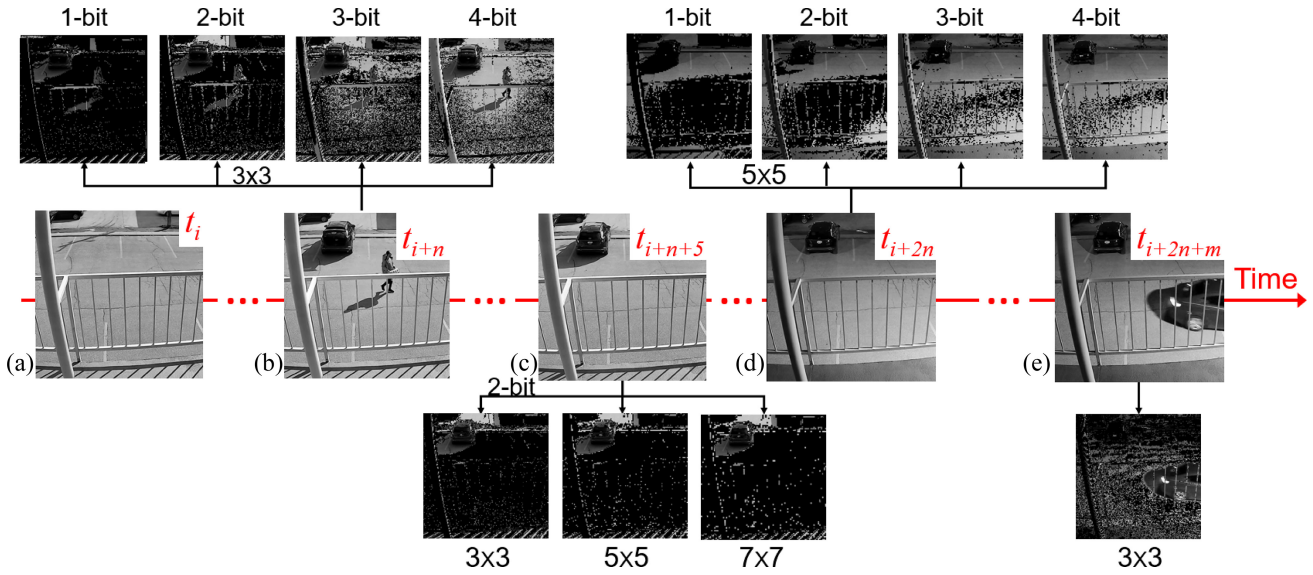

Fig. 13. Detecting object timeframes supported in the PiPSim platform. (a) → (b) Detects a car and a person leveraging different precision (1–4 bits), (a) → (c) calculates differences in the images based on different box sizes, (c) → (d) detects light variation as a new object, and (d) →(e) detects background updated and the new car detected.

Fig. 12 depicts a simple example of the procedures within Algorithm 1. In ① one 3 × 3 filter is loaded to the W_B, and the image is captured by a 9 × 9 pixel array. In the Event-Detection step ②, only the central pixels are enabled and chosen for comparison. Once the differences between the current frame and the previous (stored) one are greater than the defined threshold, based on the user's inputs, all the rows containing the pixels (the first six rows) are turned ON, as shown in Fig. 12, ③. Thus, the convolution-in-pixel operations are only performed on these rows. Several steps of convolution-in-pixel using stored weights are shown in steps ④, and ⑤. Each block of W_B is hardwired to multiple pixels, which offers highly parallel convolution-in-pixel (CiP). In Cycle 1, three output feature maps (ofmaps) are produced. In the next cycle, three other ofmaps are generated without any weights update. To implement stride, all the weights are shifted down one bit in Cycle 3 ④. After four cycles, three ofmap's columns are calculated. To slide the weights over the input, W_B shifts weights to the right (Cycle 5) and continues this process to compute all the ofmap's columns ⑤. To verify Algorithm 1, a Python script is written and executed under the conditions specified in Fig. 13, including light intensity and event occurrence. For example, in Fig. 13(a) and (b), the system finds differences between two consecutive frames, t_i and t_{i+n}, and detects an event, a car, and a person, with varied resolutions 1–4 bits. The results indicate that the algorithm works correctly without any issues. Additionally, PiPSim can calculate the total power of the system across various parameters for the algorithm, as demonstrated in the figure.

IV. Results

A. Validation Results

To validate the PiPSim model accuracy for reporting power and latency parameters, we replicated the whole architecture with peripherals in Synopsys HSPICE simulator on a computer having system configuration 128-GB RAM with AMD EPYC 7302P 16-Core Processor in SUSE Linux operating system. The state-of-the-art PiP designs are not suitable for validation as they need to provide detailed simulation parameters and configurations. Therefore, we implemented PiPA with the box size of 3 × 3 in PiPSim and SPICE at the 45-nm PTM

TABLE II
VALIDATION RESULTS FOR PIP OPERATIONS AT 45 NM

Tool	Sense Power (μW)	Sense Latency (ns)	Compute Power (μW)	Compute Latency (ns)
SPICE	6.8785e-06	2.8619e-09	1.7225e-04	1.7819e-08
PiPSim	6.44858e-06	2.9117e-09	1.728536e-04	1.82634e-08
Error Rate	**-6.6%**	**+1.7%**	**+0.35%**	**+2.4%**

TABLE III
SIMULATION TIME AND MEMORY UTILIZATION OF PIPE(II) VERSUS SPICE

Box size	(3,3)	(3,3)	(3,3)	(3,3)
Array size	32×32	64×64	128×128	256×256
PiPE(II) (s)	0.025	0.032	0.034	0.036
SPICE (s)	50.91	175.56	669.28	2639.53
speed-up	**2,036**×	**5,486**×	**19,684**×	**73,320**×
PiPE(II) (MB)	15.027	15.035	15.039	15.041
SPICE (MB)	581.88	848.57	1894.02	6217.29
memory saving	**38.72**×	**56.439**×	**125.940**×	**413.356**×

technology library [37] with the nominal $V_{DD} = 1$ V. The breakdown of results into sense and compute components are listed in Table II. We observe that all the error rates for latency and power consumption are lower than 2.4% compared with the SPICE results except for the sensing power with ∼6.6%. Notably, the delay for components is measured as FO4 (Fan-out of 4), where each component drives at most four similar elements. Therefore, it is the user's responsibility to enter a reasonable delay for additional parts. On average, PiPSim offers a model accuracy loss of 2.7%. SPICE circuit simulators provide better accuracy with higher flexibility, whereas PiPSim limits to validation of PiP-based designs. However, it offers a user-friendly platform with higher speed.

We further implement the whole architecture in SPICE and report the simulation time and memory utilization of PiPSim compared with SPICE in Table III. To investigate the impact of PiPA array size (i.e., 32×32, 64×64, 128×128, and 256×256) on the overall simulation time, we ran the simulation multiple times with a box size of 3×3. After running the application, results, including peak memory (total memory) use and total CPU time, are listed. On the other hand, the PiPE(II) procedure obtains results by estimating performance metrics rapidly using analytical models or/and predefined values provided by the user. HSPICE solves various equations, whereas, PiPSim uses precalculated components' features to compute the performance metrics, such as power consumption. We observe that PiPSim offers, on average 25 000× speed-up compared with SPICE. We can see that the larger the array size, the more speed-ups are achievable compared with SPICE. As for memory utilization, PiPSim requires, on average 158× lower memory to store and generate the results.

B. Case Studies

As a proof of concept of PiPSim, we consider the following parameters as a given configuration file, 4-D filter size $= 16 \times 1 \times 3 \times 3$, `Stride`$= 1$, and `Channel Pruning`$= 2$, where only the green channel is considered, `Quantization Level`$=$ Quinary, `Memory Technology`$=$ MRAM, `Pixel Array Size`$= 32 \times 32$, `Pixel Type`$=$ 2T, `ADC Precision`$=$ varies between 4 and 8, and `Parallelism Level`$= 3$, which means three 3×3 filters are read, simultaneously.

TABLE IV
PERFORMANCE COMPARISON OF VARIOUS SENSOR UNITS

Designs	Technology (nm)	Purpose	Frame Rate ($frame/s$)	Power (mW)	Efficiency ($TOp/s/W$)
[38]	180	2D optic flow est.	30	0.029	0.0041
[12]	180	edge*/blur/sharpen/ 1st layer CNN	480	sensing: 0.077 processing: 0.091	0.777
[8]	60/90	STP†	1000	sensing: 230 processing:363	0.386
[10]	180	1st layer BNN	1000	0.0121	1.32
[18]	65	1st layer QNN	1000	0.088	1.89
[11]	180	edge*/TMF‡	100,000	1230	0.535
[4]	180	1^{st}-layer CNN	3840	0.45 - 1.83	1.41 - 3.37
[15]	180	1^{st}-layer BNN	156	0.00014 - 0.00053	9.4-34.6
Ours	45	1st layer CNN	3000	0.00096 - 0.0028	1.37 - 4.12
Ours	180Ψ	1st layer CNN	722	0.00271 - 0.0105	0.268 - 1.25

*Edge extraction. †Spatial Temporal Processing. ‡Median Filter Threshold. ΨScale-up.

TABLE V
ACCURACY (%) COMPARISON ON CBCL FACE, SVHN, AND CIFAR-10

Configuration	bit-width	CBCL Face	SVHN	CIFAR-10
Floating-Point [39]	32-bit	98.33	96.57	91.37
MACSen [10]	1-bit (binary)	96.0	–	–
MR-PIPA [18]	2-bit (quaternary)	92.30	91.05	–
Ours	3-bit (quinary)	98.30	96.49	90.05

Table IV compares the structural and performance characteristics of recent processing-in-sensor (PiS)/processing-near-sensor (PnS)/PiP designs. While the target application and purpose behind developing each design is different as tabulated, for an impartial comparison, the power consumption of PIS units executing the similar task of processing the 1st-layer of CNN is estimated: 1) our obtained design (from now on *Ours*) and the presented accelerators in [4], [10], and [15] are the only designs supporting an entire-array computation scheme; 2) ours and the designs in [10], [11], and [15] have integrated memory components, where our design is the only design exploiting the NVMs for normally off and instant computing; 3) ours achieves a frame rate of 3000 and stands as the second fastest design supporting 1st-layer CNN computation after the design in [4] (with 3840 fps); 4) ours reduces the power consumption by 3 orders of magnitude compared to the design in [4] (i.e., the fastest 1st-layer CNN accelerator) and stands as one of the most power-efficient designs; and 5) the 1st-layer BNN accelerator in [15] shows the highest efficiency, where ours achieves 4.12 TOp/s/W. The comparison of classification accuracy between Ours and the state-of-the-art is summarized in Table V, where except the floating point (FP), all the designs quantized the 1st-layer convolution. The results show that our architecture provides higher accuracy than binary and ternary weight neural networks based on 1T and 2T-pixels. This improvement is because of five values realized by the proposed PiP design. It is worth noting that ours shows an accuracy loss of less than 1% on average compared to the FP baseline. We initially implemented and simulated our design in SPICE at the 45-nm PTM technology library. Although we did not have access to all technical configurations of the counterpart implementations, for ease of understanding, we reimplemented Ours in 180-nm technology and reported the numbers in Table IV.

V. Conclusion

In this article, we presented the first behavior-level simulation platform for the PiP system, titled PiPSim. This hierarchical structure provides flexible interfaces to customize the design at multiple levels, from device to system. In addition, a behavior-level accuracy model is proposed to estimate the accuracy of the implemented PiP architecture, which is integrated into PiPSim. PiPSim also offers several optimization steps to further accelerate the convolution step within a pixel array leveraging several procedures. The experimental results show that PiPSim reaches more than 25 000× speed-up with less than 2.5% error rate on average compared with SPICE. In the future, PiPSim will be extended to a closed-loop design exploration tool to provide the tradeoff between different designs and their estimated performance and generate the most optimized architecture concerning efficiency, throughput, accuracy, and hardware constraints.

References

[1] H. Johnson. "Digging up dark data: What puts IBM at the forefront of insight economy." 2015. [Online]. Available: https://siliconangle.com/2015/10/30/ibm-is-at-forefront-of-insight-economy-ibminsight

[2] K.-T. Tang et al., "Considerations of integrating computing-in-memory and processing-in-sensor into convolutional neural network accelerators for low-power edge devices," in *Proc. Symp. VLSI Technol.*, 2019, pp. T166–T167.

[3] A. El Gamal, D. X. Yang, and B. A. Fowler, "Pixel-level processing: Why, what, and how?" in *Proc. Sens., Cameras, Appl. Digit. Photogr.*, 1999, pp. 2–13.

[4] R. Song, K. Huang, Z. Wang, and H. Shen, "A reconfigurable convolution-in-pixel CMOS image sensor architecture," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 10, pp. 7212–7225, Oct. 2022.

[5] G. Datta et al., "A processing-in-pixel-in-memory paradigm for resource-constrained TinyML applications," *Sci. Rep.*, vol. 12, no. 1, 2022, Art. no. 14396.

[6] G. Datta et al., "P2M-DeTrack: Processing-in-pixel-in-memory for energy-efficient and real-time multi-object detection and tracking," in *Proc. IFIP/IEEE 30th Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, 2022, pp. 1–6.

[7] T.-H. Hsu et al., "AI edge devices using computing-in-memory and processing-in-sensor: From system to device," in *Proc. IEEE Int. Electron Devices Meeting IEDM*, 2019, pp. 22.5.1–22.5.4.

[8] T. Yamazaki et al., "4.9 A 1ms high-speed vision chip with 3D-stacked 140GOPS column-parallel PEs for spatio-temporal image processing," in *Proc. IEEE Int. Solid-State Circuits Conf. ISSCC*, 2017, pp. 82–83.

[9] R. LiKamWa, Y. Hou, Y. Gao, M. Polansky, and L. Zhong, "RedEye: Analog ConvNet image sensor architecture for continuous mobile vision," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. ISCA*, 2016, pp. 255–266.

[10] H. Xu et al., "MACSen: A processing-in-sensor architecture integrating MAC operations into image sensor for ultra-low-power BNN-based intelligent visual perception," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 2, pp. 627–631, Feb. 2021.

[11] S. J. Carey, A. Lopich, D. R. Barr, B. Wang, and P. Dudek, "A 100,000 fps vision sensor with embedded 535GOPS/W 256×256 SIMD processor array," in *Proc. Symp. VLSI Circuits*, 2013, pp. C182–C183.

[12] T.-H. Hsu et al., "A 0.5-V real-time computational CMOS image sensor with programmable kernel for feature extraction," *IEEE J. Solid-State Circuits*, vol. 56, no. 5, pp. 1588–1596, May 2021.

[13] L. Bose, J. Chen, S. J. Carey, P. Dudek, and W. Mayol-Cuevas, "A camera that CNNs: Towards embedded neural networks on pixel processor arrays," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1335–1344.

[14] S. Angizi, M. Morsali, S. Tabrizchi, and A. Roohi, "A near-sensor processing accelerator for approximate local binary pattern networks," *IEEE Trans. Emerg. Topics Comput.*, early access, Jun. 16, 2023, doi: 10.1109/TETC.2023.3285493.

[15] H. Xu et al., "Senputing: An ultra-low-power always-on vision perception chip featuring the deep fusion of sensing and computing," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 1, pp. 232–243, Jan. 022.

[16] S. Tabrizchi, R. Gaire, S. Angizi, and A. Roohi, "SenTer: A reconfigurable processing-in-sensor architecture enabling efficient ternary MLP," in *Proc. Great Lakes Symp. VLSI*, 2023, pp. 497–502.

[17] S. Angizi, S. Tabrizchi, D. Z. Pan, and A. Roohi, "PISA: A non-volatile processing-in-sensor accelerator for imaging systems," *IEEE Trans. Emerg. Topics Comput.*, early access, Jul. 11, 2023, doi: 10.1109/TETC.2023.3292251.

[18] M. Abedin, A. Roohi, M. Liehr, N. Cady, and S. Angizi, "MR-PIPA: An integrated multilevel RRAM (HfOx)-based processing-in-pixel accelerator," *IEEE J. Explor. Solid-State Computat. Devices Circuits*, vol. 8, no. 2, pp. 59–67, Dec. 2022.

[19] S. Tabrizchi, A. Nezhadi, S. Angizi, and A. Roohi, "AppCiP: Energy-efficient approximate convolution-in-pixel scheme for neural network acceleration," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 13, no. 1, pp. 225–236, Mar. 2023.

[20] S. Tabrizchi, S. Angizi, and A. Roohi, "TizBin: A low-power image sensor with event and object detection using efficient processing-in-pixel schemes," in *Proc. IEEE 40th Int. Conf. Comput. Design ICCD*, 2022, pp. 770–777.

[21] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, *Cacti 6.0: A Tool to Model Large Caches*, HP Lab., Palo Alto, CA, USA, Jan. 2009.

[22] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[23] M. Poremba, T. Zhang, and Y. Xie, "NVMain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems," *IEEE Comput. Archit. Lett.*, vol. 14, no. 2, pp. 140–143, Jul.–Dec. 2015.

[24] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Comput. Archit. Lett.*, vol. 15, no. 1, pp. 45–49, Jan.–Jun. 2016.

[25] L. Xia et al., "MNSIM: Simulation platform for memristor-based neuromorphic computing system," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 5, pp. 1009–1022, May 2018.

[26] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 12, pp. 3067–3080, Dec. 2018.

[27] M. J. Rasch et al., "A flexible and fast PyTorch toolkit for simulating training and inference on analog crossbar arrays," 2021, *arXiv:2104.02184*.

[28] D. Gao, D. Reis, X. S. Hu, and C. Zhuo, "Eva-CiM: A system-level performance and energy evaluation framework for computing-in-memory architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 5011–5024, Dec. 2020.

[29] N. Binkert et al., "The Gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718

[30] S. Xu, X. Chen, Y. Wang, Y. Han, X. Qian, and X. Li, "PIMSim: A flexible and detailed processing-in-memory simulator," *IEEE Comput. Archit. Lett.*, vol. 18, no. 1, pp. 6–9, Jan.–Jun. 2019.

[31] B. E. Forlin, P. Santos, A. Becker, M. Alves, and L. Carro, "Sim2PIM: A complete simulation framework for processing-in-memory," *J. Syst. Archit.*, vol. 128, Apr. 2022, Art. no. 102528.

[32] A. Banagozar et al., "CIM-SIM: Computation in memory SIMulator," in *Proc. SCOPES*, May 2019, pp. 1–4.

[33] X. Fong et al., "Spin-transfer torque devices for logic and memory: Prospects and perspectives," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 1, pp. 1–22, Jan. 2016.

[34] "DRAM power model." 2010. [Online]. Available: https://www.rambus.com/energy/

[35] J. Qiu et al., "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2016, pp. 26–35.

[36] B. Murmann. "ADC performance survey 1997–2016." 2018. [Online]. Available: http://web.stanford.edu/ murmann/adcsurvey.html

[37] W. Zhao and Y. Cao, "Predictive technology model for NANO-CMOS design exploration," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 3, no. 1, pp. 1–17, 2007.

[38] S. Park, J. Cho, K. Lee, and E. Yoon, "7.2 243.3pJ/pixel bio-inspired time-stamp-based 2D optic flow sensor for artificial compound eyes," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers ISSCC*, 2014, pp. 126–127.

[39] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1," 2016, *arXiv:1602.02830*.