

CAMJ: Enabling System-Level Energy Modeling and Architectural Exploration for In-Sensor Visual Computing

Tianrui Ma*

Washington University in St. Louis
St. Louis, MO, USA
tianrui.ma@wustl.edu

Xuan Zhang†

Washington University in St. Louis
St. Louis, MO, USA
xuan.zhang@wustl.edu

Yu Feng*

University of Rochester
Rochester, NY, USA
yfeng28@ur.rochester.edu

Yuhao Zhu†

University of Rochester
Rochester, NY, USA
yzhu@rochester.edu

ABSTRACT

CMOS Image Sensors (CIS) are fundamental to emerging visual computing applications. While conventional CIS are purely imaging devices for capturing images, increasingly CIS integrate processing capabilities such as Deep Neural Network (DNN). Computational CIS expand the architecture design space, but to date no comprehensive energy model exists. This paper proposes CAMJ, a detailed energy modeling framework that provides a component-level energy breakdown for computational CIS and is validated against nine recent CIS chips. We use CAMJ to demonstrate three use-cases that explore architectural trade-offs including computing in vs. off CIS, 2D vs. 3D-stacked CIS design, and analog vs. digital processing inside CIS. The code of CAMJ is available at: <https://github.com/horizon-research/CamJ>.

CCS CONCEPTS

• **Hardware** → **Power and energy**; • **Computer systems organization** → **Analog computers**; *Sensors and actuators*; • **General and reference** → *Estimation*.

KEYWORDS

in-sensor computing, energy modeling, analog modeling

ACM Reference Format:

Tianrui Ma, Yu Feng, Xuan Zhang, and Yuhao Zhu. 2023. CAMJ: Enabling System-Level Energy Modeling and Architectural Exploration for In-Sensor Visual Computing. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23)*, June 17–21, 2023, Orlando, FL, USA. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3579371.3589064>

*Both authors contributed equally to the paper.

†Both authors are corresponding authors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ISCA '23, June 17–21, 2023, Orlando, FL, USA

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0095-8/23/06...\$15.00

<https://doi.org/10.1145/3579371.3589064>

1 INTRODUCTION

Visual computing applications on the horizon such as autonomous machines, computational photography, and space exploration all fundamentally rely on image sensing. While conventional CMOS Image Sensors (CIS) are responsible only for “imaging”, i.e., capturing pixels, CIS increasingly integrate *computation* capabilities, ranging from signal preprocessing [12] to Deep Neural Networks (DNN) [6, 27] and spanning both the analog and digital domain. For instance, a Nikon CIS [27] integrates an image processor for per-tile exposure control; the Sony IMX 500 CIS [16] integrates a DNN accelerator with the pixel array for edge visual processing.

Computational CIS expand the traditional design space for architects and provide an exciting playground for exploring a diverse range of trade-offs. For instance, in-CIS processing consumes large volumes of pixel data *in-situ* and reduces the data transmission overhead; computing inside a sensor, however, is inefficient because CIS tend to be fabricated using older process nodes compared to standard CMOS nodes (limited by the photon sensing sensitivity [66]), which offsets the gains from reducing the communication cost. Designers also face a myriad of choices when designing the in-sensor architecture. For instance, while 3D stacking improves energy efficiency by allowing for hybrid process integration [69], it could also increase the power density and, thus, thermal-induced noise, requiring more processing downstream.

This paper presents CAMJ, a first-of-its-kind energy modeling framework that empowers designers to navigate the large algorithm-hardware co-design space. CAMJ provides a component-level energy estimation under a frame-per-second (FPS) target. To that end, CAMJ models the interplay across main structures of a computational CIS pipeline: pixel sensing → analog processing → digital processing. Thus, CAMJ enables end-to-end optimizations of the CIS architecture from photon ingestion to semantic results.

Design. As an energy modeling tool for system-level exploration, CAMJ has two design principles (Sec. 3). First, we use a declarative interface to describe the algorithm and hardware configurations. This is based on the observation that image processing algorithms have regular compute and memory access patterns; thus, a declarative description of the software and hardware is sufficient for accurate estimation of hardware access counts while simplifying users’ effort. The interface also decouples algorithm and hardware descriptions to facilitate iterative architectural explorations.

Second, we adapt the energy modeling methodology to account for the characteristics of the analog and digital domains (Sec. 4). In the digital domain, CAMJ directly asks users for the per-operation energy of a Processing Element (PE) and per-access memory energy. Tools [3, 51] and data [19, 24] are widely available to obtain these statistics, which are routinely used in today’s digital accelerator simulation [2, 18, 19, 24]. The energy of analog components, however, depends on many low-level circuit details (e.g., bias current, capacitance) that designers might not (need to) have access to, and no mature tools exist. Thus, CAMJ provides default energy models for common analog components based on designs in classic CIS [30, 34, 54, 71, 72], while exposing an interface to accept custom designs from expert users.

Validation. We perform extensive validation of CAMJ against nine recent CIS chips (Sec. 5), which cover a diverse range of process nodes, pixel types, memory sizes, and forms of processing (e.g., analog vs. digital and pixel parallel vs. column parallel). Compared to the measured *absolute* energy consumption of the chips that span several orders of magnitude, CAMJ achieves a Pearson Correlation Coefficient of 0.9999 and a Mean Absolute Percentage Error of 7.5%.

Use-cases. We use three use-cases to demonstrate the architectural explorations that CAMJ enables (Sec. 6). First, we show that a conventional 2D CIS design offers little energy benefit for computation-dominated algorithms. Second, moving toward a 3D design reduces energy due to the ability to use advanced process nodes for computation—at a cost of increasing power density. Third, analog processing in CIS could significantly reduce memory energy even when the compute energy reduction is modest.

In summary, this paper makes the following contributions:

- We provide a comprehensive survey of the design and scaling trends of CIS, pointing out new opportunities for architectural exploration.
- We propose the first component-level energy modeling tool for CIS and validate it against real silicon.
- We use CAMJ to show three use-cases, demonstrating architectural trade-offs of inside-vs-off CIS, 2D-vs-3D, and analog-vs-digital processing.

2 MOTIVATION: COMPUTATIONAL CIS DESIGN AND SCALING TRENDS

We first discuss the main design trend of CIS that underlies this paper: CIS are becoming increasingly computational (Sec. 2.1). We then explain the energy benefits of such computational CIS, a main driving force behind this design trend (Sec. 2.2). Finally, we discuss the challenges of reaping the energy benefits, which motivates this work (Sec. 2.3).

2.1 Design Trends

CIS Primer. Fundamentally, a CIS consists of two basic components as illustrated in Fig. 2a: a light-sensitive photodiode array that converts photons to charges and a read-out circuit that converts charges to digital values (i.e., raw pixels) through the analog-to-digital converters (ADC). Traditionally, raw pixels are transferred to the host, e.g., a Systems-on-a-Chip (SoC) on a smartphone, through the MIPI CSI-2 interface [22]. The Image Signal Processor (ISP) in

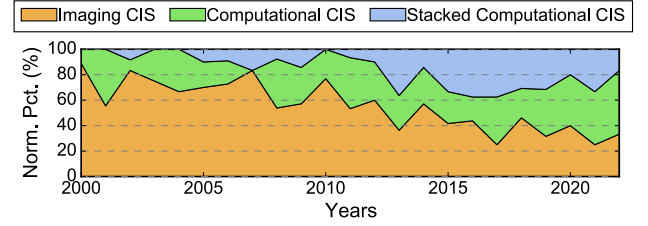


Fig. 1: Percentage of conventional CIS, computational CIS, and stacked computational CIS designs from surveying all ISSCC and IEDM papers published between Year 2000 and 2022. Increasingly more CIS designs are computational.

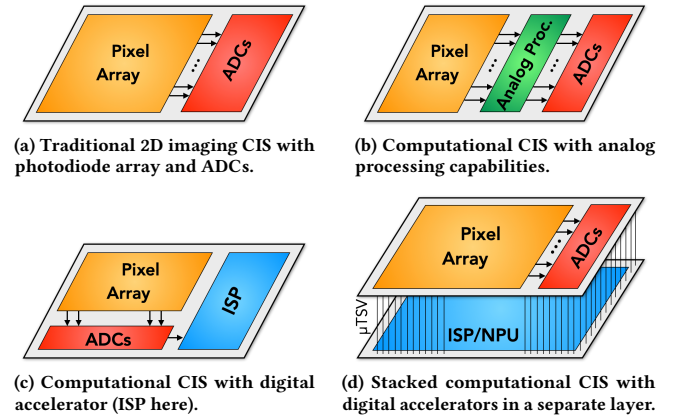


Fig. 2: CIS architecture evolution. CIS is moving away from a purely imaging device (a) to integrate computation capabilities (b)(c), sometimes in a 3D stacking fashion (d).

the SoC removes sensing artifacts (e.g., denoising) and prepares pixels for computer vision tasks and/or for visual display.

CIS Design Trend. A clear trend in CIS design is to move into the sensor computations that are traditionally carried out outside the sensor, which gives rise to the notion of *Computational CIS*. Fig. 1 shows the percentage of computational CIS papers in ISSCC and IEDM from Year 2000 and Year 2022 with respect to all the CIS papers during the same time range. Increasingly more CIS designs integrate compute capabilities.

The computations inside a CIS could take place in both the analog and the digital domain. Fig. 2b illustrates one example where analog computing is integrated into a CIS chip. Analog operations usually implement primitives for feature extraction [5, 6], object detection [72], and DNN inference [30, 70]. Fig. 2c illustrates another example that integrates digital processing, such as ISP [52], image filtering [35] and DNN [5].

As the processing capabilities become more complex, CIS design has embraced 3D stacking technologies, as is evident by the increasing number of stacked CIS in Fig. 1. Fig. 2d illustrates a typical stacked design, where the processing logic is separated from, and stacked with, the pixel array layer. The different layers communicate through hybrid bond or micro Through-Silicon Via

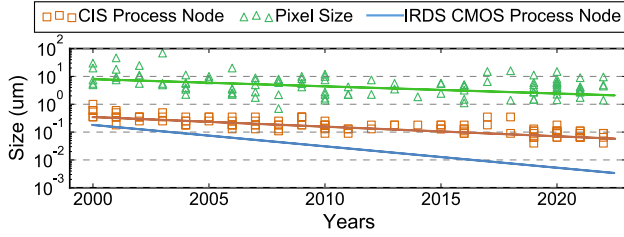


Fig. 3: CIS process node always lags behind conventional CMOS process node. This is because CIS node scaling tracks the pixel size scaling, which does not shrink aggressively due to the fundamental need of maintaining photon sensitivity.

(μ TSV) [49, 67]. The processing layer typically integrates digital processors; such as ISP [40], image processing [27, 38], and DNN accelerator [16, 45]. Three-layer stacked designs have been proposed. Sony IMX 400 [25] integrates a pixel array layer, a DRAM layer, and a digital layer with an ISP. Meta conceptualizes a three-layer design [49] with a pixel array layer, a per-pixel ADC layer, and a digital processing layer that integrates a DNN accelerator.

2.2 Benefits of Computational CIS

It is no coincidence that computational CIS emerge when energy efficiency is critical. From an architecture perspective, computational CIS provides two main energy benefits. First, moving computation inside the sensor allows the pixel data to be consumed closer to where they are generated. Doing so reduces the data transmission energy, which could dominate the overall energy consumption.

Specifically, data communication inside a CIS using a μ TSV consumes about 1 pJ/B, whereas the energy cost of transmitting one Byte out of the CIS through the MIPI CSI-2 interface consumes about 100 pJ of energy [49]. As an example, if a CIS is capable of executing an object detection DNN directly, the data volume that has to be transmitted out of the sensor is simply a few Bytes (object location and label), as opposed to, say, 6 MB, for a 1080p image.

Second, computational CIS also provides a natural platform for analog acceleration, since the pixel data originate from the analog domain to begin with, obviating the need for energy-intensive digital to analog converters that often dominate the hardware overheads in conventional analog accelerators. Compare to digital processing, analog processing minimizes energy-intensive data conversion [8, 50] and can reduce both the computation and memory energy consumption.

2.3 Challenges and Design Space

Moving computation inside a CIS, however, is not without challenges. Most importantly, processing inside the sensor is far less efficient than that outside the sensor, fundamentally because the CIS process node significantly lags behind that of the conventional CMOS. Fig. 3 illustrates this difference, where square markers show the process nodes used in CIS designs from all ISSCC papers appeared during Year 2000 and Year 2022, which include leading industry CIS designs at different times. We overlay a trend line regressed from these CIS designs to better illustrate the scaling trend. As a comparison, the blue line at the bottom represents the conventional

CMOS technology node scaling laid out by International Roadmap for Devices and Systems (IRDS) [1].

At around Year 2000, the CIS process node started lagging behind that of the conventional CMOS node, and the gap is increasing. CIS design today commonly use 65nm and older process nodes. This gap is not an artifact of the CIS designs we pick; it is fundamental: there is simply no need to aggressively scale down the process node because the pixel size does not shrink much. The triangles in Fig. 3 represent the pixel sizes of all the CIS designs we surveyed. The slope of CIS process node scaling almost follows exactly that of the pixel size scaling. The reason that pixel size does not shrink is to ensure light sensitivity: a small pixel reduces the number of photons it can collect, which directly reduces the dynamic range and the Signal-to-Noise ratio (SNR) [4].

Inefficient in-sensor processing can be mitigated through 3D stacking technologies [69], which allows for heterogeneous integration: the pixel layer and the computing layer(s) can use their respective, optimal process node. Stacking, however, could increase power density especially when future CIS integrate more processing capabilities. Therefore, harnessing the power of (stacked) computational CIS requires exploring a large design space and address key challenges, some of which we list below. Providing a tool to easily navigate the design space is the goal of our CAMJ framework.

- Whether and what to compute in vs. off CIS?
- How to architect each layer in stacked CIS to achieve energy reduction without increasing power density?
- What to compute in the analog vs. digital domains?

3 CAMJ FRAMEWORK

No framework to date allows designers to explore the complicated design space of computational CIS at a system level. Our CAMJ framework is designed to fill this void (Sec. 3.1). We first outline the design principles of CAMJ, followed by an overview of the CAMJ design internals (Sec. 3.2). We use a concrete example to demonstrate from a designer perspective how CAMJ is used (Sec. 3.3).

3.1 When is CAMJ Used in the Design Cycle?

CAMJ is meant to be used for *system-level* exploration after each component design is sketched out; an analogy would be Systems-on-a-Chip (SoC) vs. accelerator design. Before system-level exploration, a team usually has at hand a range of component-level designs, which could be licensed Intellectual Property (IP) blocks, reference designs from the literature, or earlier designs from other teams in the organization (e.g., using a synthesis flow or High-Level Synthesis tools); in all cases the component-level energy behavior is known or can be modeled using external tools like Aladdin [63] and OpenRAM [23].

CAMJ helps designers make design decisions when assembling the individual (digital and analog) components into an optimal system. Ideally, a designer uses CAMJ to estimate the system energy given initial designs of individual components; using the estimation, a designer can *iteratively* refine the components/system design. For instance, CAMJ can identify energy bottlenecks and guide the re-design of corresponding components. Orthogonally, a designer can use CAMJ to explore optimal mapping and partitioning of the

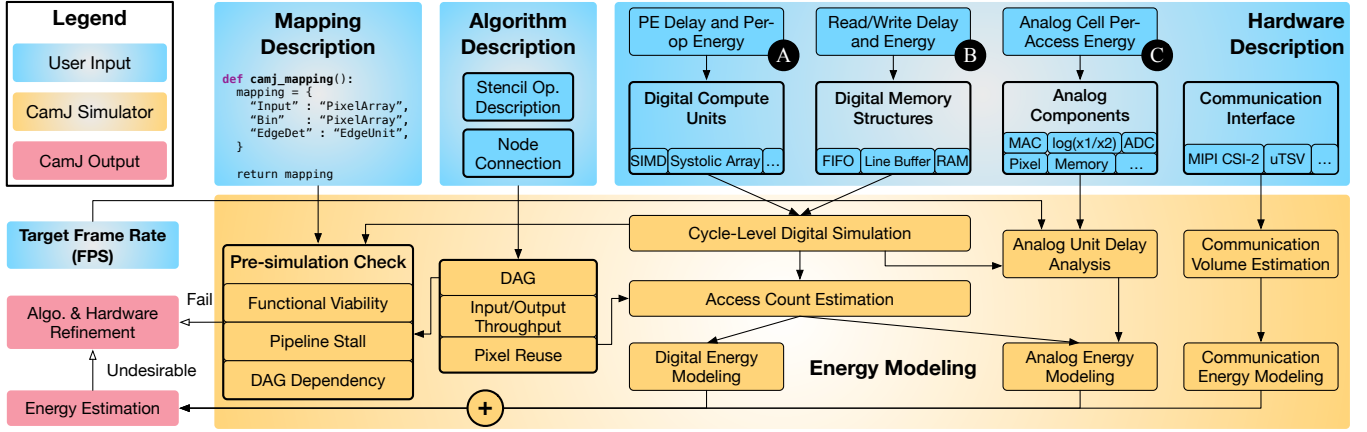


Fig. 4: Overview of the CAMJ framework, which provides a component-level energy estimation under a target frame rate (FPS). Users provide an algorithm and hardware description and the mapping between the two. A, B, and C denote the per-access energy of digital computation, digital memory, and analog units, respectively; the first two are obtained from external tools while the last one is provided within CAMJ. See Tbl. 1 for a list of digital compute/memory and analog components supported in CAMJ. CAMJ then estimates the access count to each hardware component to obtain the overall energy estimation.

algorithms between analog vs. digital domains or in vs. off CIS to minimize overall system energy under performance targets.

CAMJ is *not* a synthesis tool; it does not generate (nor estimate the energy of) an accelerator. Rather, CAMJ can be used in conjunction with HLS: one could use HLS to first generate an accelerator and then use CAMJ to explore, in the bigger system, how/whether that accelerator would fit in a computational CIS to maximize end-to-end application gains.

3.2 Design Principles and Overview

As with any energy modeling tool [7, 33, 43], the total CIS energy is sum of the product of 1) the access count to each hardware unit and 2) the per-access energy consumption. Therefore, the central objective of CAMJ is to develop a modeling methodology that accurately estimates those two statistics using a programming interface that, critically, only requires user inputs for information that cannot be automatically inferred. Fig. 4 shows an overview of the CAMJ framework. Sec. 3.3 and Sec. 4 discuss the programming interface and modeling details, respectively. Here, we provide an overview of the interface and modeling methodology.

Interface. CAMJ observes that CIS deal with stencil-based image processing, which has regular computation and memory access patterns (i.e., little to no control flow) that are statically mapped to hardware units. The access count statistics can, thus, be inferred with only the high-level algorithm description and hardware configuration without knowing the implementation details. Therefore, CAMJ exposes a *declarative* interface, in which an algorithm is described by a DAG and the input/output/stencil dimensions at each node and the hardware is abstracted as a set of basic units, each performing a specific sensing, computation, or memory operation.

CAMJ’s interface also *decouples* the description of an algorithm, the underlying hardware, and the mapping between the two. A decoupled interface facilitates an iterative system design process, during which algorithm, hardware, and algorithm to hardware mapping can change independently. For instance, one can evaluate

algorithmic changes by re-writing the algorithm description without touching the hardware design, or explore different algorithm-to-hardware mappings (e.g., split between analog vs. digital and between in vs. off sensor) by describing a new mapping.

Internal Modeling. CAMJ judiciously uses different methodologies to obtain the per access energy in the digital vs. analog domains. For digital structures, CAMJ directly asks users to provide the per-cycle energy of computation PEs (A) and the per-access energy of memory units (B). These statistics are usually obtained by an ASIC synthesis flow or from commonly used tools (e.g., CACTI [3] and OpenRAM [23]), and are routinely used in today’s digital accelerator simulators for energy estimation [2, 19, 37].

One might be surprised to find that CAMJ directly asks for the per-cycle/access energy of the digital structures. This is because of the design philosophy of CAMJ (Sec. 3.1): it is *not* used to generate digital accelerators; rather, it helps assess how an accelerator fits in the entire computational CIS system. For that reason, CAMJ expects designers to have a preliminary design of the digital accelerators (whether it’s your manual design, HLS generated, or a licensed IP), in which case one will have the per-cycle/access energy statistics.

Unlike digital structures, few energy modeling tool exists for analog structures, whose energy consumption (C) depend on many low-level circuit details (e.g., load capacitance, gain, bias current) that are cumbersome and perhaps unreasonable to ask system-level designers for. Our design decision is to expose a low-level interface to accept these parameters from expert users, but also provide default energy models based on classic implementations of analog components and delay analyses of these components (Sec. 4.1).

Finally, CAMJ performs a series of pre-simulation design checks to ensure that the algorithm and hardware combination 1) is functionally viable (e.g., ADCs must exist between the analog and digital domain), 2) does not have pipeline stalls (to avoid accumulating long frame latency), and 3) has well-formed dependencies in the algorithm DAG (e.g., no circle). We provide feedback upon check



Fig. 5: An example of defining a simple CIS using the CAMJ programming interface in Python. The hardware architecture of the simulated CIS is illustrate at the top.

failures and a detailed energy breakdown, which helps designers iteratively refine the algorithm and/or hardware.

3.3 Programming Interface

We use a running example in Fig. 5 to introduce the programming interface and illustrate the main design decisions.

An Example. The Python code in Fig. 5 shows a concrete example to use the programming interface of CAMJ. In this conceptual CIS design with a 32×32 pixel array, every 2×2 pixel tile is first averaged (i.e., “binned”) to produce a 16×16 image. The sensor then performs a digital edge detection on the image before sending the edge data through the MIPI CSI interface. The `camj_hw_config`

Table 1: A list of hardware units supported in CAMJ. APS: Active Pixel Sensor; DPS: Digital Pixel Sensor; PWM: Pulse Width Modulation; MAC: Multiply-Accumulate.

	Analog (A-COMPONENT)	Digital
Memory	Passive/Active, Sample-and-Hold	FIFO, Line Buffer, Double-Buffered SRAM
Compute	Pixel (APS, DPS, PWM), ADC, MAC, Max, Scaling, Add, log, Abs, Comparator	Systolic Array, Generic Pipelined Accelerator

function and the `camj_sw_config` function describe the hardware components and the algorithm, respectively. The `camj_mapping` maps each algorithm stage to a hardware component. We explain each part next.

Algorithm Description. The code in `camj_sw_config` describes the DAG of the entire processing pipeline, starting from the raw pixels generated by the pixel array (PixelInput), which go through two processing stages: `bin_stage` for pixel binning and `edge_stage` for edge detection. The `set_input_stage` method connects the stages together to form a DAG.

Notice how the algorithm description does not require the actual arithmetic details; we observe that image processing algorithms can be abstracted as stencil operations that operate on a local window of pixels at a time [26, 58] — convolution (or image filtering in conventional image processing parlance) being a prime example. This observation holds in all the ISSCC/IEDM papers since Year 2000 we surveyed. Irregular computations complicate hardware design and increase energy, defeating the purpose of in-CIS computing.

Therefore, users express only the input/output image dimensions (`input_size`, `output_size`) along with the stencil window (`kernel`) and stride size (`stride`). Given the regular computation and data access pattern of stencil operations, CAMJ could accurately estimate the access counts to different hardware structures for energy estimations. Nonetheless, CAMJ does accept as input a memory trace offline collected for an irregular algorithm, which can then be integrated with external tools such as DRAMPower [10] to estimate the energy consumption for irregular algorithms.

Hardware Description. `camj_hw_config` describes the hardware architecture, which we illustrate at the top of Fig. 5. The hardware description consists of two components: analog processing units and digital processing units.

① **Analog Units.** CIS hardware necessarily starts from analog units, which, at a high level, are described as a set of Analog Functional Arrays (AFA), which is in turn is composed of a set of Analog Functional Components (A-COMPONENTS). The most important AFA in a CIS is the pixel array (`pixel_array`), in which each A-COMPONENT is a pixel, which is added to the pixel array through the `add_component` method. In the example of Fig. 5, the pixel array is followed by another AFA, i.e., the ADC array (`adc_array`), where each A-COMPONENT is an ADC.

From users’ perspective, each A-COMPONENT performs a particular kind of (arithmetic) operation. In addition to a pixel or an ADC, CAMJ provides other common A-COMPONENTS used in CIS such as MAC or logarithmic operations. The complete list of analog

A-COMPONENTS is in Tbl. 1. The energy consumption of each A-COMPONENT, which is dictated by its circuit-level implementation, is abstracted away from the users by the `impl` method. Sec. 4.2 will later describe how we model the energy of each A-COMPONENT by mapping it to its analog circuit implementation.

What users do have to provide, however, is the signal dimension (`num_input` and `num_output`) and signal domain (`input_domain` and `output_domain`) of an AFA's input and output data. These parameters allow CAMJ to check whether the simulated CIS is functionally viable. Specifically, the `input_domain` of a consumer unit and the `output_domain` of a producer unit must match. If, for instance, the producer is in the charge domain and the consumer is in the voltage domain, CAMJ will ask designers to insert a charge-to-voltage conversion component¹, which has energy implications. Similarly, if the `num_input` of a consumer unit and the `num_output` of a producer unit do not match, the hardware must have an analog buffer in-between, which, again, could have energy implications.

② **Digital Units.** The digital part of the hardware is described by specifying a set of compute units that communicate through memory structures. In this example, the compute unit is the edge detection accelerator (instantiated through `ComputeUnit`), which reads from the line buffer (`LineBuffer`), a pre-defined memory structure, that stores data from the pixel array, an analog unit as described before.

Column 2 of Tbl. 1 lists the memory structures and compute units available in CAMJ. We support three memory structures commonly found in image/vision processing: FIFO (`FIFO`), line buffer [[26](#), [68](#)] (`LineBuffer`), and double-buffered SRAM (`DoubleBuffer`). The compute units are abstracted as pipelined accelerators through the `ComputeUnit` interface. We also provide a `SystolicArray` class to describe a systolic array due to its importance in executing DNNs.

With the generic pipelined accelerator interface (`ComputeUnit`), users can model a wide range of (image processing) accelerators. To describe a pipelined accelerator, CAMJ requires three main parameters: the shape of pixels read per cycle (`input_pixel_per_cycle`), the shape of pixels generated per cycle (`output_pixel_per_cycle`), and the pipeline depth (`num_stages`). Using these statistics, CAMJ performs cycle-level simulation for two purposes. First, CAMJ can check whether the accelerator will stall the CIS pipeline and, if so, asks for a re-design of the accelerator. Second, CAMJ can estimate the total latency of the digital domain, which is critical for analog energy estimation. Stall checking and latency estimation are critical for analog energy estimation as we will discuss in Sec. 4.1.

Mapping. The `camj_mapping` function maps each algorithm stage to a hardware unit. The code is self-evident. Users can simply remap the algorithm to hardware to explore a different system design. The decoupling of algorithm and hardware description through the mapping function also enables easy expression of hardware reuse—by simply mapping different algorithm nodes to a hardware component.

¹unless the output of the consumer is in the voltage domain, where the inherent capacitor of the consumer naturally acts as an analog buffer.

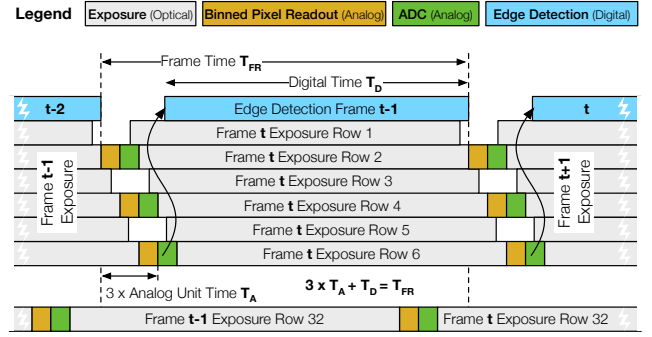


Fig. 6: The (not-to-scale) pipeline diagram of the example in Fig. 5 when there is no pipeline stall.

4 ENERGY MODELING METHODOLOGY

The energy consumption per frame of a CIS sensor is the sum of that of the analog, digital, and data communication:

$$E^{\text{frame}} = E_a^{\text{frame}} + E_d^{\text{frame}} + E_c^{\text{frame}} \quad (1)$$

Before we describe how the analog component E_a^{frame} (Sec. 4.2), digital component E_d^{frame} (Sec. 4.3), and communication component E_c^{frame} (Sec. 4.4) are modeled separately, we first discuss a prerequisite of energy modeling: delay estimation (Sec. 4.1).

4.1 Delay Estimation

The energy consumption, both analog and digital, is correlated with the circuit speed. For example, in the analog domain an operational amplifier (OpAmp) with higher response speed requires larger bias current, which increase the energy consumption (assuming the OpAmp is active over a fixed duration, e.g., when used for an analog frame buffer). The latency of digital units is estimated through cycle-level simulation as described in the previous section. The delay of an analog unit, in contrast, depends on many parameters specific to a fully-designed circuit. We find it cumbersome and error-prone to ask users for input: users often find themselves tuning low-level parameters only to end up with a design that misses the target frame rate. Instead, CAMJ's insight is that each analog unit's delay can be automatically inferred from the prescribed frame rate.

Specifically, the fundamental observation that CAMJ relies on is that the CIS pipeline is designed to never stall. This is because the input data to the pipeline is generated at a constant rate as the pixel array is exposed to light at the constant speed. If the pipeline ever stalls in a later stage, the frame latency would gradually accumulate, leading to excessively long responsive latency or frame drops. Therefore, CIS designers ensure that the hardware pipeline never stalls. In a fully-pipelined hardware, each pipeline stage must have roughly the same delay; this is the basis of our delay estimation.

Example. Fig. 6 shows the pipeline timing for the example in Fig. 5. The frame time T_{FR} is 1/FPS, where FPS is the target frame rate. In the diagram, the frame time is the delay between when the pixels of the current frame can be read-out to when the computation of the current frame finishes. The “Binned Pixel Readout” and “ADC” are the two analog units, who share the same delay T_A (i.e., balanced pipeline) to be estimated. The “Edge Detection” is the digital unit, which starts once the second line has been written to the line buffer.

To estimate T_A , we first simulate the digital domain to estimate the latency of the entire digital domain (T_D here). Given the frame time T_{FR} , we can then estimate how much time is left for the analog units. In this example, $T_A = \frac{T_{FR}-T_D}{3}$.

CAMJ will analyze the hardware description and, upon detecting potential stalls, asks the user to re-design the hardware to avoid stall. Specifically, CAMJ checks to avoid three scenarios: 1) pixel required is not generated by the producer yet, 2) the memory in-between two stages is full, or 3) the number of access ports in the memory structures is not enough.

4.2 Analog Energy Modeling

The analog energy per frame, $E_{\text{analog}}^{\text{frame}}$, is the sum of the energy consumption per access of each A-COMPONENT weighted by the access count to that component. Refer to Tbl. 1 for a list of A-COMPONENTS that CAMJ supports.

$$E_a^{\text{frame}} = \sum_i (E_a^{\text{component}[i]} \times \text{Num}_{\text{access}}^{\text{component}[i]}) \quad (2)$$

Modeling A-COMPONENTS Access Count. The access count to a A-COMPONENT is the number of times the A-COMPONENT is used per frame. Recall from Sec. 3.3 that each A-COMPONENT is part of an Analog Functional Array (AFA). CAMJ leverages the fundamental observation that stencil operations in image processing have regular computation and memory access patterns and, thus, the access count to each A-COMPONENT in the same AFA is the same.

As a result, the access count of a component i is simply the ratio between the total number of operations mapped to the AFA j that contains the component ($\text{Num}_{\text{ops}}^{\text{AFA}[j]}$) and the number of components in that AFA ($\text{Num}_{\text{component}[i]}^{\text{AFA}[j]}$):

$$\text{Num}_{\text{access}}^{\text{component}[i]} = \frac{\text{Num}_{\text{ops}}^{\text{AFA}[j]}}{\text{Num}_{\text{component}[i]}^{\text{AFA}[j]}} \quad (3)$$

The numerator is easily derived from the algorithm description of a stencil operation (e.g., calculating the number of MAC operations in a convolution). The denominator is the `num_component` attribute of the AFA (see Fig. 5).

Modeling A-COMPONENTS Access Energy. Internally, each A-COMPONENT is built from a set of analog cells, which we call A-CELLS. Modeling per-access energy of an A-COMPONENT requires knowing its cell-level implementation. Expert users can define new cell parameters and/or cell-level implementation of an A-COMPONENT. Absent those, each A-COMPONENT has a default implementation, surveyed from classic and recent CIS designs [30, 34, 54, 71, 72]. For instance, a 4T-APS pixel A-COMPONENT consists of a photodiode (PD) A-CELL, a floating diffusion node (FD) A-CELL, and a source follower (SF) A-CELL; a multiplier implemented by switched-capacitor charge re-distribution [42] consists of a capacitor array A-CELL and an OpAmp A-CELL.

We now describe our energy modeling of A-COMPONENTS, but keep in mind that these design details are abstracted away from typical users. $E_a^{\text{component}}$ is the weighted sum of the energy consumption of each constituting A-CELL in the component and the

access counts to the A-CELL:

$$E_a^{\text{component}[i]} = \sum_j E_a^{\text{cell}[j]} \times \text{Num}_{\text{access}}^{\text{cell}[j]} \quad (4)$$

Despite large varieties of high-level analog circuits, the A-CELL used for analog in-sensor computing can be categorized to three classes according to circuit characteristics: dynamic A-CELL, static-biased A-CELL, and non-linear A-CELL. They each consume energy in a different way.

① **Dynamic A-CELL.** The energy of a dynamic circuit comes from the charging and discharging of the total capacitance in the circuit:

$$E_a^{\text{cell,d}} = \sum_i N_c C[i] \times V_{\text{VS}}[i]^2 \quad (5)$$

where N_c represents the total number of capacitance nodes in the dynamic circuit, and $C[i]$ and $V_{\text{VS}}[i]$ are the capacitance and the voltage swing at i^{th} capacitance node, respectively. Typical dynamic A-CELLS include capacitive digital-to-analog converter (CDAC) and passive analog memory.

In Equ. 5, V_{VS} is determined by the analog supply V_{DDA} and the number of transistors placed between the analog supply and the ground. The nodal capacitance C is determined by its thermal noise and the computation precision. To guarantee the accuracy of analog computing, the maximum thermal noise should be kept below $\frac{1}{2}$ LSB of the data resolution:

$$\sigma_{\text{thermal}} = \sqrt{\frac{kT}{C}}, \quad 3\sigma_{\text{thermal}} < \frac{1}{2} \text{LSB} \quad (6)$$

where $\text{LSB} = V_{\text{VS}}/2^{\text{data resolution}}$. Data resolution is algorithm dependent. For example, if $V_{\text{VS}} = 1\text{V}$ and the required resolution is 8-bit, the thermal noise should be less than $\frac{1}{3} \frac{1}{2} \frac{1\text{V}}{2^8} = 2.6\text{mV}$, from which C is obtained.

② **Static-biased A-CELL.** The energy of a static-biased circuit comes from the integration of the bias current over a specific time period under the analog supply V_{DDA} :

$$E_a^{\text{cell,s}} = V_{\text{DDA}} \times I_{\text{bias}} \times t_{\text{static}} \quad (7)$$

where I_{bias} is the bias current and t_{static} is the time during which the A-CELL is statically biased.

We provide two ways to estimate I_{static} based on circuit details. For A-CELLS where I_{bias} directly drives the load capacitance (e.g. static-biased SF in a pixel), I_{bias} is determined by charging up the load within the given time:

$$I_{\text{bias,1}} = \frac{C_{\text{load}} \times V_{\text{VS}}}{t_{\text{static}}} \quad (8)$$

where C_{load} is the load capacitance. The energy is reduced to:

$$E_a^{\text{cell,s}} = C_{\text{load}} \times V_{\text{VS}} \times V_{\text{DDA}} \quad (9)$$

For A-CELLS where I_{bias} does not directly drive the load capacitance (e.g. differential operational amplifier in analog memory or discrete-time integrator), I_{bias} can be determined by the classic $\frac{g_m}{I_d}$ method [32]:

$$I_{\text{bias,2}} = \frac{2\pi \times C_{\text{load}} \times \text{GBW}}{g_m/I_d} \quad (10)$$

where $\frac{g_m}{I_d}$ is a technology-insensitive factor ranging from 10 to 20 depending on the inversion level of the transistors, and GBW is product between gain (G) and bandwidth (BW).

To use Equ. 7 and Equ. 10, CAMJ must estimate BW and t_{static} , both of which depend on the A-CELL delay. Specifically, BW is the reciprocal of the A-CELL delay and t_{static} is:

$$t_{\text{static}} = T_A - \sum_i^K t_i^{\text{cell}} \quad (11)$$

where T_A is the delay of the A-COMPONENT containing the A-CELL and is estimated in Sec. 4.1; K is the number of cells before the current A-CELL on the A-COMPONENT critical path, and t_i^{cell} is the delay of an A-CELL. Absent timing condition from users, we evenly allocate the A-COMPONENT delay to each A-CELL, based on the fact that the analog signal uni-directionally flows through the A-COMPONENTS we support so all A-CELLS are on the critical path.

③ **Non-linear A-CELL.** For those circuits with non-linear transfer functions, such as ADCs and comparators (which are essentially 1-bit ADCs), they contain both dynamic/static-biased circuit cells and digital logic so it is difficult to estimate the energy from analytical formulas. Instead, we use the ADC's Walden Figure-of-Merit (FoM) plot [53] surveyed from recently published CIS papers, which shows the ADC's energy-per-conversion vs. its sampling rate. Specifically, given the ADC sampling rate (the reciprocal of the A-CELL delay) we, absent detailed user input, use the median energy-per-conversion at that sampling rate as the estimation. The total energy of non-linear A-CELL is thus obtained by the product of its estimated FoM and the number of required conversions:

$$E_a^{\text{cell, nl}} = \text{FoM [J/conversion]} \times \text{Num}_{\text{conversion}} \quad (12)$$

The access counts to a specific A-CELL are the number of times the A-CELL is used along both the spatial and temporal scale to generate one A-COMPONENT output:

$$\text{Num}_{\text{access}}^{\text{cell}} = \text{Num}_{\text{spatial}}^{\text{cell}} \times \text{Num}_{\text{temporal}}^{\text{cell}} \quad (13)$$

For example, if an A-CELL represents a static-biased SF in a pixel, $\text{Num}_{\text{spatial}}^{\text{static-SF}}$ would be the number of SFs in the pixel and $\text{Num}_{\text{temporal}}^{\text{static-SF}}$ would be the number of times the pixel charge is read out (e.g., 2 if correlated double sampling is used to reduce noise [9]). The access counts information for A-CELLS is hard-coded for each A-COMPONENT and is abstracted away from typical users, but can be updated for a custom design.

4.3 Digital Energy Modeling

The digital energy of a frame E_d^{frame} is the sum of the computation energy of each compute unit E_d^{comp} and the energy of each memory structure E_d^{mem} :

$$E_d^{\text{frame}} = \sum_i E_d^{\text{comp}}[i] + \sum_j E_d^{\text{mem}}[j] \quad (14)$$

The energy of each compute unit is the product of the energy per cycle E_d^{cycle} and the number of cycles $\text{Num}_{\text{cycle}}$:

$$E_d^{\text{comp}}[i] = E_d^{\text{cycle}}[i] \times \text{Num}_{\text{cycle}} \quad (15)$$

We rely on users to provide E_d^{cycle} , which usually is obtained through HLS/ASIC synthesis flows. The cycle counts, in contrast, are obtained through cycle-level simulation by CAMJ.

The energy consumption for memory accesses is the sum of leakage energy and the dynamic access energy; the latter is the product of the energy consumption for one memory read E_d^{read} or write (E_d^{write}) and the total number of memory reads (Num_{read}) or writes ($\text{Num}_{\text{write}}$):

$$E_d^{\text{mem}}[j] = E_d^{\text{read}}[j] \times \text{Num}_{\text{read}}[j] + E_d^{\text{write}}[j] \times \text{Num}_{\text{write}}[j] + P_d^{\text{leakage}}[j] \times \frac{1}{\text{FR}} \times \alpha \quad (16)$$

The leakage energy is the product of the leakage power P_d^{leakage} and the memory active time (i.e., not power-gated), which is a fraction α of the frame time $\frac{1}{\text{FR}}$. Users supply the dynamic read/write energy and leakage power; the access counts and the active time are from the CAMJ simulation.

4.4 Communication Energy Modeling

The communication power is dominated by the energy to transfer the data outside the sensor using the energy-hungry MIPI CSI-2 interface and, in the case of 3D-stacking CIS, the energy of μTSV . In literature the energy of the two interfaces is usually given for energy per Byte. Therefore, the communication energy is given by:

$$E_c^{\text{frame}} = E_c^{\text{mipi}} \times \text{Num}_{\text{Bytes}}^{\text{mipi}} + E_c^{\text{tsv}} \times \text{Num}_{\text{Bytes}}^{\text{tsv}} \quad (17)$$

E_c^{mipi} and E_c^{tsv} are user supplied with represented data reported in the literature [48]. The data volume statistics in both interfaces are generated in CAMJ simulation (based on the algorithm description and algorithm to hardware mapping).

5 CAMJ VALIDATION

In this section, we validate CAMJ against real measurement data from nine recent CIS chips [5, 13, 16, 29, 30, 54, 61, 70, 72] shown in Table 2. These designs span a range of design dimensions including 2D and 3D designs, different process nodes, pixel types, as well as PE designs and memory sizes in the analog and digital domains.

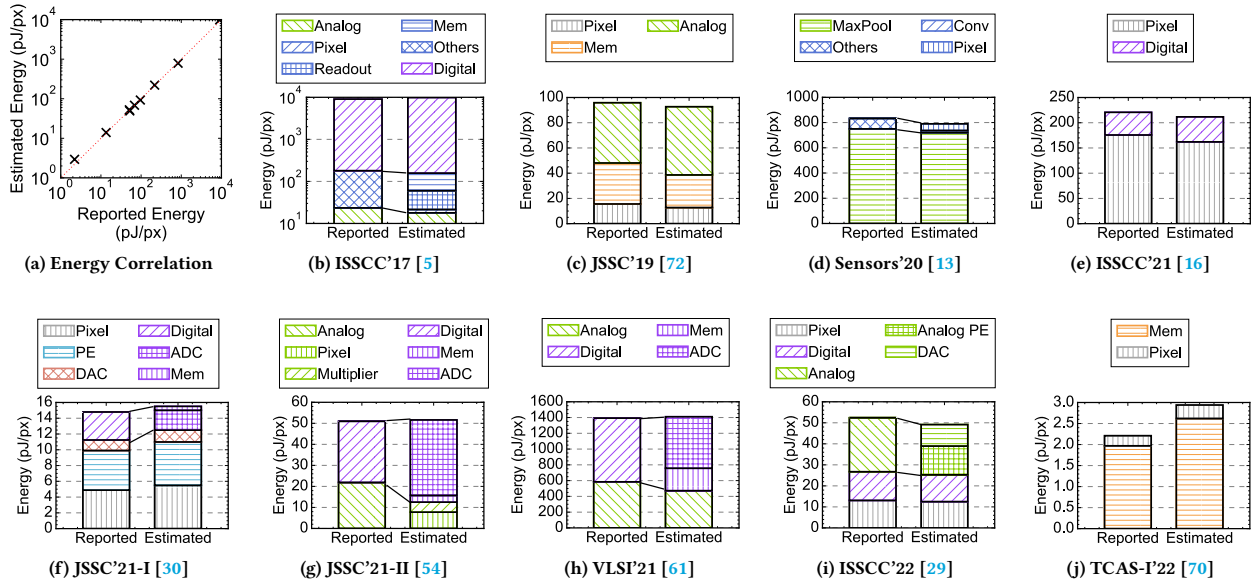
Fig. 7 compares the estimated and actual energy per pixel reported in the original papers. Our estimations closely match the measured results, which span several orders of magnitude, showing both the diversity of the CIS design styles and the wide system power/energy scale that CAMJ can flexibly support and accurately model. Across all designs, CAMJ achieves a Mean Absolute Percentage Error of 7.5% and a Pearson Correlation Coefficient of 0.9999.

Fig. 7b – Fig. 7j compare the detailed energy breakdown across the nine designs. Whenever possible, we use the circuit parameters reported in the papers. For SRAMs, we use DESTINY [57] to obtain per-access energy. The three papers that perform digital computation all execute DNNs, where a PE is a MAC unit; we use the synthesis result of a 65 nm MAC unit design for per-MAC energy [5], and scale it to other process nodes based on classic CMOS scaling [60, 64].

While overall CAMJ provides an accurate component-level and full-system energy estimation, we find two key reasons behind result mismatches. First, the results are less accurate when CAMJ

Table 2: Summary of CIS designs for validation, which cover a wide range of design variations. *indicates data not reported in the original papers and are based on our educated guess. Unit of analog memory size is “number of analog values”.

CIS	Process Node	Stacked	Analog					Digital	
			Pixel	Memory	PE Operation	PE Position	Op Domain	Memory	PE Size
ISSCC'17 [5]	65nm	No	3T APS	20 × 80	Avg&Add	Column&Chip	Charge&Voltage	160KB	4 × 4 × 64
JSSC'19 [72]	130nm	No	4T APS	4 × 240	Logarithmic Sub.	Column	Voltage	-	-
Sensors'20 [13]	110nm	No	4T APS	No	MAC&MaxPool	Column	Voltage	-	-
ISSCC'21 [16]	65nm/22nm	Yes	4T APS*	No	-	-	-	8MB	1 × 2304
JSSC'21-I [30]	180nm	No	PWM	No	MAC	Column	Time&Current	-	-
JSSC'21-II [54]	110nm	No	4T APS	No	MAC	Column	Charge	-	-
VLSI'21 [61]	65nm/28nm	Yes	DPS	No	-	-	-	6MB	-
ISSCC'22 [29]	180nm	No	PWM	No	MAC	Column	Time&Current	256B*	1
TCAS-I'22 [70]	180nm	No	3T APS	No	Mul.&Add	Pixel&Chip	Current	-	-

**Fig. 7: Validation results. CAMJ achieves a Pearson correlation coefficient of 0.9999. Several papers lump different components into the coarse-grained “Analog”, “Digital”, or “Others” categories. We show detailed breakdown and indicate when the sum of several fine-grained categories in our estimation corresponds to a coarse-grained category in the original papers.**

does not have access to detailed design parameters. For example, the pixel estimation in Fig. 7f, Fig. 7g, and Fig. 7j shows an absolute error of 12.4%, 38.9%, and 33.3%, respectively, due to insufficient circuit parameters on pixel ramp-generator (Fig. 7f), pixel parasitic capacitance (Fig. 7g), and photodiode voltage swing (Fig. 7j). Similarly, the analog PE in Fig. 7f and Fig. 7b shows an absolute error of 9.3% and 23.7% due to insufficient circuit parameters on sampling capacitance (Fig. 7f) and sense amplifier conversion energy (Fig. 7b), respectively. In contrast for Fig. 7c, where the detailed design parameters are provided for the analog PE, the estimation error is only 0.4%.

The other source of inaccuracy comes from the mismatch between the actual circuit design and CAMJ’s default circuit template. For example, the ADCs in Fig. 7g and Fig. 7h show an absolute difference of 31.7% and 16%, respectively²; the original designs use low-power dynamic technique (Fig. 7g) whereas CAMJ estimates

the energy of ADC based on the FOM survey [53]. The memory in Fig. 7j shows an estimation error of 33.0% because the original design uses customized 8T SRAMs while CAMJ uses standard 6T SRAMs from DESTINY [57], resulting in higher leakage power.

6 ARCHITECTURAL EXPLORATION

We now demonstrate three complementary examples of using CAMJ to explore architectural trade-offs. The three use-cases complement each other. First, we explore when moving computation inside the CIS brings energy benefits (Sec. 6.1). We show that the energy reduction is small in a conventional 2D design. We then show that introducing 3D stacking, which allows for hybrid integration, further improve the energy efficiency at a cost of higher power density (Sec. 6.2). Finally, we discuss the trade-off of moving computation into the analog domain (Sec. 6.3).

²Both papers consider ADC as a digital unit, which is what we use here.

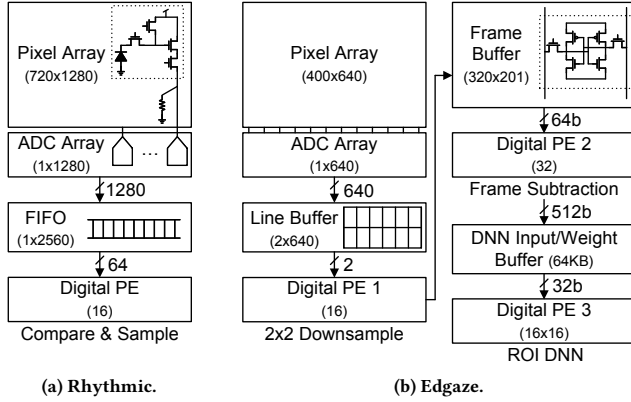
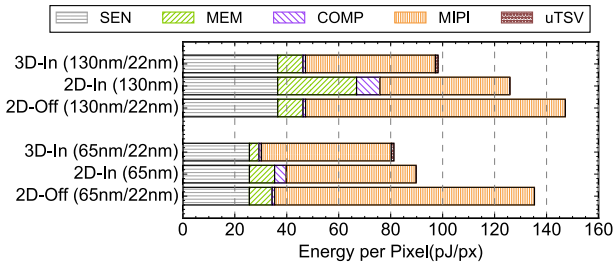
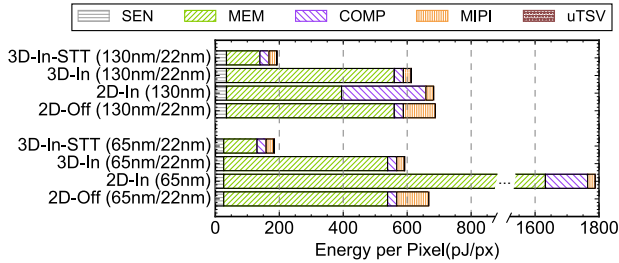


Fig. 8: Hardware design for (a) Rhythmic Pixel Regions [37], which is a ROI-based image encoder and (b) Ed-Gaze, which is a gaze tracking algorithm. In their original designs, everything after ADC takes place off CIS; we evaluate the benefits of bringing the entire execution into the CIS.



(a) Results on Rhythmic Pixel Regions.



(b) Results on Ed-Gaze.

Fig. 9: Energy consumption comparison between 2D in-sensor (2D-In), 2D off-sensor (2D-Off), 3D in-sensor (3D-In), and 3D in-sensor with STT-RAM (3D-In-STT) processing. SEN: everything up to and including ADCs; MEM: memory; COMP: computation; MIPI: MIPI CSI-2 communication; uTSV: μ TSV communication between layers.

6.1 Computing Inside vs. Off Sensor

Computing inside CIS reduces the data transmission cost by consuming pixel data inside the sensor. To explore the benefits, we evaluate two recent papers, Rhythmic Pixel Regions [37] and Ed-Gaze [17], both of which generate a small Region of Interest (ROI)

from the original full-resolution image and, thus, can potentially benefit from in-sensor computing by moving the ROI generation inside the sensor.

Fig. 8a and Fig. 8b illustrate the execution flow of the two algorithms, respectively. Everything after the ADC is executed outside the sensor in the original papers, and we use CAMJ to evaluate the energy consumption after moving the entire execution inside the sensor. Specifically:

- **Rhythmic Pixel Regions:** a 1280×720 pixel array is processed by a dedicated accelerator (Compare & Sample) to generate the ROI, which on average reduces the image size by 50%. The ROI generation performs roughly 7.4×10^6 arithmetic operations per frame.
- **Ed-Gaze:** a 640×400 pixel array is first downsampled by 2×2 , and then processed by an pixel-wise subtraction operation with respect to the previous frame to generate an event map, which is then processed by a DNN to generate the ROI. The ROI, on average, reduces the image size by 25%. The DNN dominates the computation and performs about 5.76×10^7 MAC operations per frame.

We use CAMJ to evaluate two hardware configurations:

- **2D-In (H):** a 2D CIS fabricated in the H process node; the entire execution is performed inside the CIS.
- **2D-Off (H/L):** a 2D CIS fabricated in the H process node; everything after the ADC takes place on an SoC, which is fabricated in the L process node. We evaluate two CIS process nodes, 130 nm and 65 nm, both common in CIS designs (Tbl. 2). We set the SoC process node to 22 nm.

Fig. 9a shows the energy of Rhythmic Pixel Region under different designs. Overall, 2D-In reduces the energy compared to 2D-Off. When CIS process node is 130nm, 2D-In saves 14.5% compared to 2D-Off. This saving improves to 33.4% when the CIS process node is 65nm. The energy reduction comes from reducing the amount of data (full-resolution image vs. ROI) that has to be communicated through the MIPI CIS-2 interface (MIPI in the figure). This data communication cost reduction comes at the expense of increasing the computation (COMP) and memory access (MEM) cost, both of which increase because of the older process node in the CIS.

Ed-Gaze's results are shown in Fig. 9b, which tells a different story. Computing inside the CIS ends up consuming much more energy than computing on the SoC. This is because communication cost is light for Ed-Gaze: the communication cost contributes to only 15.0% of the total energy in the off-sensor system to begin with. Thus, the additional energy costs of computation and memory accesses inside the sensor far outweigh the reduction in communication cost. Interestingly, 65nm 2D-In consumes more energy than its 130nm counterpart, because the 65nm node is known to have high leakage power [20]. In Ed-Gaze, the frame buffer must be on during the entire frame time without power gating, since a frame must always be retained in order for frame subtraction.

Finding 1: *Computation inside CIS is less efficient than that off CIS. In-CIS computing saves energy only when the process node of the CIS is not too far behind that of the SoC and the application is communication-dominant.*

Table 3: Power Density. Unit is mW/mm².

Process Node Pixel/Compute	Rhythmic Pixel Regions			Ed-Gaze		
	2D-Off	2D-In	3D-In	2D-Off	2D-In	3D-In
130nm/22nm	0.05	0.09	0.06	0.19	0.30	0.78
65nm/22nm	0.03	0.05	0.04	0.11	2.24	0.70

6.2 Comparison of 2D CIS and 3D-stacked CIS

We use CAMJ to explore the benefits of 3D stacked CIS. We use the same algorithms in Rhythmic Pixel Regions and Ed-Gaze and consider two additional 3D configurations:

- **3D-In (H/L):** a two-layer stacked CIS, where the pixel layer is fabricated in the **H** process node and the compute layer is fabricated in an advanced process node **L**. All post-ADC operations take place in the compute layer.
- **3D-In-STT (H/L):** similar to **3D-In (H/L)** except the SRAM in the compute layer is replaced with a STT-RAM, which we model using NVMEExplore [55]. Rhythmic Pixel Regions lacks STT-RAM results, because it requires only a 2K memory, which NVMEExplore does not support.

Comparing **3D-In** and **2D-In** in Fig. 9a shows that 3D integration reduces the energy by 15.8% on average for Rhythmic Pixel Regions. This is because the digital PEs and the SRAMs now use the same process node as that of the SoC; moving computation inside the sensor no longer increases the computation and memory energy but still enjoys the significant data volume reduction. The additional cost of moving data via μ TSV is insignificant, due to the low energy cost of μ TSV.

The energy reduction from 3D integration is even higher for Ed-Gaze. Comparing **3D-In** and **2D-In** in Fig. 9b, 3D stacking reduces the energy by 38.5% on average, because memory energy contributes to 71.3% of the total energy in **2D-In**, which is significantly reduced when using a 22nm node.

That said, the memory energy still dominates in **3D-In**, because the frame buffer cannot be power-gated as explained before, consuming non-trivial leakage power. To further reduce the memory energy, we explore STT-RAM, which is known to have low leakage. By replacing SRAM with STT-RAM, **3D-In-STT** further reduces the overall energy by 69.1% and 68.5% compared to **3D-In** under the 65nm/22nm and the 130nm/22nm combination, respectively.

Power Density. While 3D stacking is known to increase power density, we use CAMJ to show that the impact on power density is algorithm-specific. We use a conservative area estimation to obtain a power density upper bound while leaving a comprehensive area modeling to future work. Specifically, we use the pixel array to approximate the analog area and use SRAM area to approximate the digital area.

Tbl. 3 compares the power density across the three sensor variants for the two algorithms. The power density of Rhythmic Pixel Regions shows no significant difference among the three variants, because the power of Rhythmic Pixel Regions is dominated by communication, which is not greatly affected by stacking. In contrast, the power density of Ed-Gaze more than doubles by 3D stacking under the 130nm/22nm combination. This is because **3D-In** is 3 $\times$ smaller in area than **2D-In**. Under the 65nm/22nm combination,

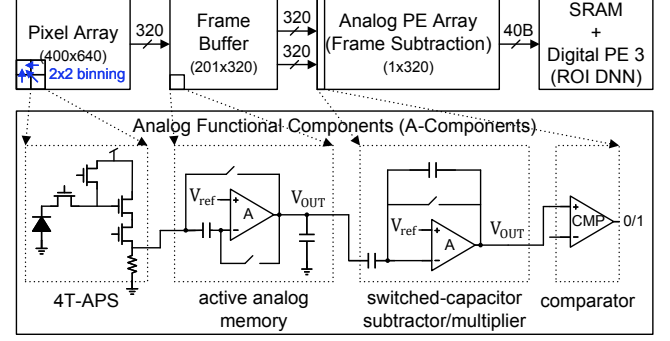


Fig. 10: Mixed-signal CIS design for Ed-Gaze. We move the first digital stages in Fig. 8b to the analog domain.

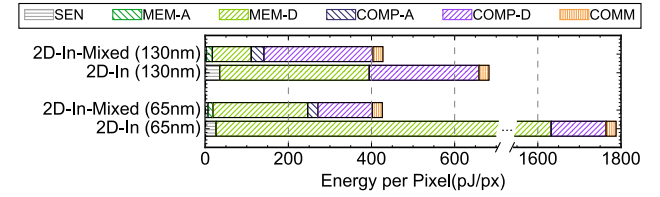


Fig. 11: Energy comparison between mixed-signal in-sensor computation and fully-digital in-sensor computation on Ed-Gaze. COMP/MEM-D: digital compute and memory; COMP/MEM-A: analog compute and memory.

2D-In has higher power density due to the high leakage power of 65nm process, which is avoided in the stacked design.

It is worth noting that the *absolute* power density of 3D-stacked sensors is still very low, in fact three to four orders of magnitude lower than the power density of typical CPUs (up to 1W/mm² [14]) and GPUs (up to 0.3W/mm² [11]). Such a low power density will unlikely lead to thermal hotspots and create a cooling challenge [73]. However, higher power density increases the thermal-induced noise and worsens the imaging and computing quality [36]. End-to-end application optimization must take into account the noise impact, an exploration that CAMJ enables and that we leave to future work.

Finding 2: 3D stacking saves energy but increases power density — for compute-dominant applications. The absolute power density is not high enough to create thermal hotspots but could increase noise, warranting further studies.

6.3 Comparison of Analog and Digital Computing

We use CAMJ to explore the benefits of in-sensor analog computing. In particular, we use Ed-Gaze for the case study and consider a mixed-signal configuration:

- **2D-In-Mixed:** a 2D CIS, where the first two stages in the algorithm (Fig. 8b), 2×2 downsampling and frame subtraction, are implemented in analog while last stage (ROI DNN) is implemented in the digital domain.

Fig. 10 shows how the Ed-Gaze is mapped to a mixed-signal CIS. Inside the pixel array, the 2×2 downsampling is done through

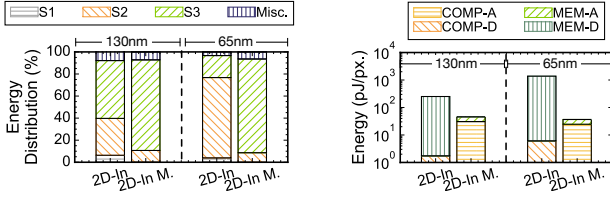


Fig. 12: Normalized energy breakdown among the three first two stages (S1, S2, S3).

pixel binning (similar to that in Fig. 5). The analog frame buffer stores the downsampled analog pixel values, which are read by an analog PE array for frame subtraction. Each analog PE consists of a switched-capacitor subtractor/multiplier for absolute subtraction and a comparator for frame delta digitization. The output of the Analog PE array enters the SRAM array, at which point the hardware is the same as that in **2D-In**. For a fair comparison and to ensure area overhead is well accounted for, we conservatively set all the capacitors to 100fF. Despite the over-sizing, analog design still yields at least 27% less area than the digital counterpart.

Fig. 11 compares **2D-In-Mixed** and **2D-In**. Moving the first stages of the Ed-Gaze algorithm to the analog domain reduces the energy by 38.8% and 77.1%. The energy reduction comes from two sources: removing the ADCs (indicated by lower SEN) and replacing SRAMs in the first two stages with analog buffers (indicated by lower MEM-D). The reduction in MEM-D is particularly significant for the 65nm node, where the SRAM leakage power is high. To corroborate the results, Fig. 12 shows the normalized energy breakdown among the three stages (S1, S2, and S3). S3 (DNN) becomes the dominant stage after moving first two stages into analog domain, showing the effectiveness of analog processing.

Interestingly, the energy reduction is obtained when the compute energy of the first two stages slightly increase. Fig. 13 shows the energy breakdown of the first two stages. While the memory energy reduces, the compute energy increases in the mixed-signal mode. This is because to maintain an 8-bit precision the OpAmp consumes too much energy (Equ. 6). A caveat is that the analog design presented here, which uses active switched-capacitor circuits, is based on our specific implementation choice. It is conceivable that different designs would yield different efficiency results.

Finding 3: While analog computing is known for reducing ADC and computing energy, the energy saving is also attributed to lower analog memory energy, especially for memory-intensive applications, which many in-CIS use-cases are.

7 RELATED WORK

Emerging CIS designs are discussed in Sec. 2 and Sec. 5.

Power Modeling. Power/energy modeling is a cornerstone of architectural exploration. Prior power models of CPUs [7, 44, 62], GPUs [28, 33, 43], and memory [3, 23, 55, 57] have enabled a plethora of power/energy optimizations. Fundamentally, CAMJ shares the same, bottom-up modeling methodology, where energy is estimated from access counts and per-access energy. Additionally, CAMJ provides a clean programming interface to integrate

other architectural simulators [18, 19, 59, 63] and memory modeling tools [55, 57] to model bespoke accelerators and memories.

Prior analog power modeling requires either detailed transistor-level parameters [65] or is based on the statistic models of particular analog circuits [41]. Lim et al. [47] decomposes a mixed-signal circuit into basic cells and accelerate the mixed-signal simulation by approximating the transfer function of each cell. CAMJ uses a similar decomposition methodology but specifically targets CIS.

CIS Modeling. No comprehensive CIS modeling framework exists. Two recent papers from Meta use first-order analytical model to estimate the energy of their custom CIS design, i.e., 3D stacking with DPS [21, 48]. It does not provide the level of flexibility to accommodate general CIS design and architecture exploration as supported by CAMJ.

LiKamWa et al. [46] provide a coarse-grained CIS power model using the idle and active period/power without considering the hardware implementation details. CAMJ, instead, models the hardware with finer granularity to achieve finer-grained architectural exploration. Kodukula et al. [36] cite coarse-grained component energy of typical CIS designs and builds a thermal model. CAMJ, can provide more accurate power/energy modeling that feeds into such thermal model.

Visual Computing Optimizations. Recent work discusses the possibility of processing inside an CIS to reduce the data transmission cost, e.g., Ed-Gaze [17], Rhythmic Pixel Regions [37], Reid et al. [56], and SplitNets [15]. All, however, rely on first-order energy models. Using CAMJ, we study the algorithms in Ed-Gaze and Rhythmic Pixel Regions and quantify their benefits (Sec. 6). Many recent visual computing optimizations use motion vectors that can be naturally generated during imaging to simplify downstream vision processing [18, 74]. It is interesting to explore how motion estimation can be integrated into the CIS using CAMJ. CAMJ can also be integrated with visual computing benchmarks [31, 39] to study in-CIS computing for different workloads.

8 CONCLUSION

CAMJ is a silicon-validated, component-level energy modeling and architecture exploration framework for computational CIS. It accepts high-level descriptions of image processing algorithms and hardware designs, and enables exploration of architectural trade-offs, e.g., in vs. off CIS, 2D vs. 3D design, and analog vs. digital processing. CAMJ is the first step in the promising field of in-sensor visual computing.

REFERENCES

- [1] [n. d.]. International Roadmap for Devices and Systems. <https://irids.ieee.org/>.
- [2] Vahideh Akhlaghi, Amir Yazdanbakhsh, Kambiz Samadi, Rajesh K Gupta, and Hadi Esmailzadeh. 2018. Snapea: Predictive early activation for reducing computation in deep convolutional neural networks. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 662–673.
- [3] Rajeev Balasubramanian, Andrew B Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New tools for interconnect exploration in innovative off-chip memories. *ACM Transactions on Architecture and Code Optimization (TACO)* 14, 2 (2017), 1–25.
- [4] M Bigas, Enric Cabruja, Josep Forest, and Joaquim Salvi. 2006. Review of CMOS image sensors. *Microelectronics journal* 37, 5 (2006), 433–451.
- [5] Kyeongryeol Bong, Sungpill Choi, Changhyeon Kim, Donghyeon Han, and Hoi-Jun Yoo. 2017. A low-power convolutional neural network face recognition processor and a CIS integrated with always-on face detector. *IEEE Journal of Solid-State Circuits* 53, 1 (2017), 115–123.

- [6] Kyeongryeol Bong, Sungpill Choi, Changhyeon Kim, Sanghoon Kang, Youchang Kim, and Hoi-Jun Yoo. 2017. 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, 248–249.
- [7] David Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. Wattch: A framework for architectural-level power analysis and optimizations. *ACM SIGARCH Computer Architecture News* 28, 2 (2000), 83–94.
- [8] Weidong Cao, Yilong Zhao, Adith Boloor, Yinhe Han, Xuan Zhang, and Li Jiang. 2022. Neural-PIM: Efficient Processing-In-Memory With Neural Approximation of Peripherals. *IEEE Trans. Comput.* 71, 9 (2022), 2142–2155.
- [9] Raffaele Capocchia, Assim Boukhayma, and Christian Enz. 2020. Experimental Verification of the Impact of Analog CMS on CIS Readout Noise. *IEEE Transactions on Circuits and Systems I: Regular Papers* 67, 3 (2020), 774–784.
- [10] Karthik Chandrasekar, Christian Weis, Yonghui Li, Benny Akesson, Norbert Wehn, and Kees Goossens. 2012. DRAMPower: Open-source DRAM power & energy estimation tool. URL: <http://www.drampower.info> 22 (2012).
- [11] John Y Chen. 2009. GPU technology trends and future requirements. In *2009 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 1–6.
- [12] Chih-Chi Cheng, Chia-Hua Lin, Chung-Te Li, and Liang-Gee Chen. 2008. iVisual: An intelligent visual sensor SoC with 2790 fps CMOS image sensor and 205 GOPS/W vision processor. *IEEE Journal of Solid-State Circuits* 44, 1 (2008), 127–135.
- [13] Jaihyuk Choi, Sungjae Lee, Youngdoo Son, and Soo Youn Kim. 2020. Design of an always-on image sensor using an analog lightweight convolutional neural network. *Sensors* 20, 11 (2020), 3101.
- [14] Andrew Danowitz, Kyle Kelley, James Mao, John P Stevenson, and Mark Horowitz. 2012. CPU DB: recording microprocessor history. *Commun. ACM* 55, 4 (2012), 55–63.
- [15] Xin Dong, Barbara De Salvo, Meng Li, Chiao Liu, Zhongnan Qu, HT Kung, and Ziyun Li. 2022. SplitNets: Designing Neural Architectures for Efficient Distributed Computing on Head-Mounted Systems. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12559–12569.
- [16] Ryoji Eki, Satoshi Yamada, Hiroyuki Ozawa, Hitoshi Kai, Kazuyuki Okuike, Hareesh Gowtham, Hidetomo Nakanishi, Edan Almog, Yoel Livne, Gadi Yuval, et al. 2021. 9.6 A 1/2.3 inch 12.3 Mpixel with on-chip 4.97 TOPS/W CNN processor back-illuminated stacked CMOS image sensor. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 64. IEEE, 154–156.
- [17] Yu Feng, Nathan Goulding-Hotta, Asif Khan, Hans Reysenrover, and Yuhao Zhu. 2022. Real-Time Gaze Tracking with Event-Driven Eye Segmentation. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE, 399–408.
- [18] Yu Feng, Paul Whatmough, and Yuhao Zhu. 2019. Asv: Accelerated stereo vision system. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 643–656.
- [19] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. Tetris: Scalable and efficient neural network acceleration with 3d memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. 751–764.
- [20] Georges Gielen and Wim Dehaene. 2005. Analog and digital circuit design in 65 nm CMOS: End of the road?. In *Design, Automation and Test in Europe*. IEEE, 37–42.
- [21] Jorge Gomez, Saavan Patel, Syed Shakib Sarwar, Ziyun Li, Raffaele Capocchia, Zhao Wang, Reid Pinkham, Andrew Berkovich, Tsung-Hsun Tsai, Barbara De Salvo, et al. 2022. Distributed On-Sensor Compute System for AR/VR Devices: A Semi-Analytical Simulation Framework for Power Estimation. *arXiv preprint arXiv:2203.07474* (2022).
- [22] Camare Working Group. 2021. MIPI White Paper: An Introductory Guide to MIPI Automotive SerDes Solutions (MASS). <https://www.mipi.org/introductory-guide-to-mass>
- [23] Matthew R Guthaus, James E Stine, Samira Ataei, Brian Chen, Bin Wu, and Mehdi Sarwar. 2016. OpenRAM: An open-source memory compiler. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–6.
- [24] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 243–254.
- [25] Tsutomu Haruta, Tsutomu Nakajima, Jun Hashizume, Taku Umebayashi, Hiroshi Takahashi, Kazuo Taniguchi, Masami Kuroda, Hiroshi Sumihoro, Koji Enoki, Takatsugu Yamasaki, et al. 2017. 4.6 A 1/2.3 inch 20Mpixel 3-layer stacked CMOS Image Sensor with DRAM. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 76–77.
- [26] James Hegarty, John Brunhaver, Zachary DeVito, Jonathan Ragan-Kelley, Noy Cohen, Steven Bell, Artem Vasilyev, Mark Horowitz, and Pat Hanrahan. 2014. Darkroom: compiling high-level image processing code into hardware pipelines. *ACM Trans. Graph.* 33, 4 (2014), 144–1.
- [27] Tomoki Hirata, Hironobu Murata, Hideaki Matsuda, Yojiro Tezuka, and Shiro Tsunai. 2021. 7.8 A 1-inch 17Mpixel 1000fps Block-Controlled Coded-Exposure Back-Illuminated Stacked CMOS Image Sensor for Computational Imaging and Adaptive Dynamic Range Control. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 64. IEEE, 120–122.
- [28] Sunpyo Hong and Hyesoon Kim. 2010. An integrated GPU power and performance model. In *Proceedings of the 37th annual international symposium on Computer architecture*. 280–289.
- [29] Tzu-Hsiang Hsu, Guan-Cheng Chen, Yi-Ren Chen, Chung-Chuan Lo, Ren-Shuo Liu, Meng-Fan Chang, Kea-Tiong Tang, and Chih-Cheng Hsieh. 2022. A 0.8 V Intelligent Vision Sensor with Tiny Convolutional Neural Network and Programmable Weights Using Mixed-Mode Processing-in-Sensor Technique for Image Classification. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 65. IEEE, 1–3.
- [30] Tzu-Hsiang Hsu, Yi-Ren Chen, Ren-Shuo Liu, Chung-Chuan Lo, Kea-Tiong Tang, Meng-Fan Chang, and Chih-Cheng Hsieh. 2020. A 0.5-V real-time computational CMOS image sensor with programmable kernel for feature extraction. *IEEE Journal of Solid-State Circuits* 56, 5 (2020), 1588–1596.
- [31] Muhammad Huzaifa, Rishi Desai, Samuel Grayson, Xutao Jiang, Ying Jing, Jae Lee, Fang Lu, Yihan Pang, Joseph Ravichandran, Finn Sinclair, et al. 2021. IL-LIXR: Enabling End-to-End Extended Reality Research. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 24–38.
- [32] Paul G Jespers. 2010. *The gm/ID Methodology, a sizing tool for low-voltage analog CMOS Circuits*. Springer.
- [33] Vijay Kandiah, Scott Peverelle, Mahmoud Khairy, Junrui Pan, Amogh Manjunath, Timothy G Rogers, Tor M Aamodt, and Nikos Hardavellas. 2021. AccelWattch: A Power Modeling Framework for Modern GPUs. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 738–753.
- [34] Amandeep Kaur, Deepak Mishra, KM Amogh, and Mukul Sarkar. 2020. On-array compressive acquisition in cmos image sensors using accumulated spatial gradients. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 2 (2020), 523–532.
- [35] Seong-Jin Kim, Kwang-Hyun Lee, Sang-Wook Han, and Euisik Yoon. 2005. A 200/spl times/160 pixel CMOS fingerprint recognition SoC with adaptable column-parallel processors. In *ISSCC. 2005 IEEE International Digest of Technical Papers. Solid-State Circuits Conference, 2005*. IEEE, 250–256.
- [36] Venkatesh Kodukula, Saad Katrawala, Britton Jones, Carole-Jean Wu, and Robert LiKamWa. 2021. Dynamic temperature management of near-sensor processing for energy-efficient high-fidelity imaging. *Sensors* 21, 3 (2021), 926.
- [37] Venkatesh Kodukula, Alexander Shearer, Van Nguyen, Srinivas Lingutla, Yifei Liu, and Robert LiKamWa. 2021. Rhythmic pixel regions: multi-resolution visual sensing system towards high-precision visual computing at low power. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 573–586.
- [38] Oichi Kumagai, Atsumi Niwa, Katsuhiko Hanzawa, Hidetaka Kato, Shinichiro Futami, Toshio Ohyama, Tsutomu Imoto, Masahiko Nakamizo, Hirotaka Murakami, Tatsuki Nishino, et al. 2018. A 1/4-inch 3.9 Mpixel low-power event-driven back-illuminated stacked CMOS image sensor. In *2018 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 86–88.
- [39] Hyoukjun Kwon, Krishnakumar Nair, Jamin Seo, Jason Yik, Debabrata Mohapatra, Dongyuan Zhan, Jinook Song, Peter Capak, Peizhao Zhang, Peter Vajda, et al. 2022. XRBench: An Extended Reality (XR) Machine Learning Benchmark Suite for the Metaverse. *arXiv preprint arXiv:2211.08675* (2022).
- [40] Minh Kwon, Seunghyun Lim, Hyeokjong Lee, Il-Seon Ha, Moo-Young Kim, Il-Jin Seo, Suho Lee, Yongsuk Choi, Kyunghoon Kim, Hansoo Lee, et al. 2020. A Low-Power 65/14nm Stacked CMOS Image Sensor. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–4.
- [41] E. Lauwers and G. Gielen. 2002. Power estimation methods for analog circuits for architectural exploration of integrated systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 10, 2 (2002), 155–162.
- [42] Edward H. Lee and S. Simon Wong. 2017. Analysis and Design of a Passive Switched-Capacitor Matrix Multiplier for Approximate Computing. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 261–271.
- [43] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M Aamodt, and Vijay Janapa Reddi. 2013. GPUWattch: Enabling energy optimizations in GPGPUs. *ACM SIGARCH Computer Architecture News* 41, 3 (2013), 487–498.
- [44] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd annual IEEE/ACM international symposium on microarchitecture*. 469–480.
- [45] Robert LiKamWa, Yunhui Hou, Julian Gao, Mia Polansky, and Lin Zhong. 2016. Redeye: analog convnet image sensor architecture for continuous mobile vision. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 255–266.
- [46] Robert LiKamWa, Bodhi Priyantha, Matthai Philipose, Lin Zhong, and Paramvir Bahl. 2013. Energy characterization and optimization of image sensing toward continuous mobile vision. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. 69–82.

- [47] Byong Chan Lim and Mark Horowitz. 2019. An Analog Model Template Library: Simplifying Chip-Level, Mixed-Signal Design Verification. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 1 (2019), 193–204.
- [48] Chiao Liu, Andrew Berkovich, Song Chen, Hans Reyserhove, Syed Shakib Sarwar, and Tsung-Hsun Tsai. 2019. Intelligent vision systems—bringing human-machine interface to AR/VR. In *2019 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 10–5.
- [49] Chiao Liu, Song Chen, Tsung-Hsun Tsai, Barbara De Salvo, and Jorge Gomez. 2022. Augmented Reality-The Next Frontier of Image Sensors and Compute Systems. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. IEEE, 426–428.
- [50] Tianrui Ma, Weidong Cao, Fei Qiao, Ayan Chakrabarti, and Xuan Zhang. 2022. HOGEye: neural approximation of hog feature extraction in rram-based 3d-stacked image sensors. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. 1–6.
- [51] Sparsh Mittal, Rujia Wang, and Jeffrey Vetter. 2017. DESTINY: A comprehensive tool with 3D and multi-level cell memory modeling capability. *Journal of Low Power Electronics and Applications* 7, 3 (2017), 23.
- [52] Hirotaka Murakami, Eric Bohannon, John Childs, Grace Gui, Eric Moule, Kat-suhiko Hanzawa, Tomofumi Koda, Chiaki Takano, Toshimasa Shimizu, Yuki Takizawa, et al. 2022. A 4.9 Mpixel Programmable-Resolution Multi-Purpose CMOS Image Sensor for Computer Vision. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. IEEE, 104–106.
- [53] B. Murmann. [n.d.]. ADC Performance Survey 1997–2022. <http://web.stanford.edu/~murmman/adcsurvey.html>.
- [54] Chanmin Park, Wenda Zhao, Injun Park, Nan Sun, and Youngcheol Chae. 2021. A 51-pj/pixel 33.7-dB PSNR 4× compressive CMOS image sensor with column-parallel single-shot compressive sensing. *IEEE Journal of Solid-State Circuits* 56, 8 (2021), 2503–2515.
- [55] Lillian Pentecost, Alexander Hankin, Marco Donato, Mark Hempstead, Gu-Yeon Wei, and David Brooks. 2022. NVMEExplorer: A Framework for Cross-Stack Comparisons of Embedded Non-Volatile Memories. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 938–956.
- [56] Reid Pinkham, Tanner Schmidt, and Andrew Berkovich. 2020. Algorithm-aware neural network based image compression for high-speed imaging. In *2020 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*. IEEE, 196–199.
- [57] Matt Poremba, Sparsh Mittal, Dong Li, Jeffrey S Vetter, and Yuan Xie. 2015. Destiny: A tool for modeling emerging 3d nvm and edram caches. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1543–1546.
- [58] Wajahat Qadeer, Rehan Hameed, Ofer Shacham, Preethi Venkatesan, Christos Kozyrakis, and Mark A Horowitz. 2013. Convolution engine: balancing efficiency & flexibility in specialized computing. In *Proceedings of the 40th IEEE Annual International Symposium on Computer Architecture*.
- [59] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883* (2018).
- [60] Satyabrata Sarangi and Bevan Baas. 2021. DeepScaleTool: A tool for the accurate estimation of technology scaling in the deep-submicron era. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.
- [61] Min-Woong Seo, Myunglae Chu, Hyun-Yong Jung, Suksan Kim, Jiyou Song, Junan Lee, Sung-Yong Kim, Jongyeon Lee, Sung-Jae Byun, Daehee Bae, Minkyung Kim, Gwi-Deok Lee, Heesung Shim, Changyong Um, Changhwa Kim, In-Gyu Baek, Doowon Kwon, Hongki Kim, Hyuksoon Choi, Jonghyun Go, JungChak Ahn, Jaekyu Lee, Changrok Moon, Kyupil Lee, and Hyoung-Sub Kim. 2021. A 2.6 e-rms Low-Random-Noise, 116.2 mW Low-Power 2-Mp Global Shutter CMOS Image Sensor with Pixel-Level ADC and In-Pixel Memory. In *2021 Symposium on VLSI Circuits*. 1–2.
- [62] Yakun Sophia Shao and David Brooks. 2013. Energy characterization and instruction-level energy model of Intel’s Xeon Phi processor. In *International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 389–394.
- [63] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2014. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 97–108.
- [64] Aaron Stillmaker and Bevan Baas. 2017. Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm. *Integration* 58 (2017), 74–81.
- [65] Christer Svensson and J Jacob Wikner. 2010. Power consumption of analog circuits: a tutorial. *Analog Integrated Circuits and Signal Processing* 65, 2 (2010), 171–184.
- [66] Albert Theuvsen. 2021. 1.4 There’s More to the Picture Than Meets the Eye*, and in the future it will only become more so. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. IEEE, 30–35.
- [67] H. Tsugawa, H. Takahashi, R. Nakamura, T. Umebayashi, T. Ogita, H. Okano, K. Iwase, H. Kawashima, T. Yamasaki, D. Yoneyama, J. Hashizume, T. Nakajima, K. Murata, Y. Kanaishi, K. Ikeda, K. Tatani, T. Nagano, H. Nakayama, T. Haruta, and T. Nomoto. 2017. Pixel/DRAM/logic 3-layer stacked CMOS image sensor technology. In *2017 IEEE International Electron Devices Meeting (IEDM)*. 3.2.1–3.2.4.
- [68] Paul N Whatmough, Chuteng Zhou, Patrick Hansen, Shreyas Kolala Venkata-ramanaiah, Jae-sun Seo, and Matthew Mattina. 2019. Fixynn: Efficient hardware for mobile computer vision via transfer learning. *arXiv preprint arXiv:1902.11128* (2019).
- [69] Yuan Xie and Jishen Zhao. 2015. Die-stacking architecture. *Synthesis Lectures on Computer Architecture* 10, 2 (2015), 1–127.
- [70] Han Xu, Ningchao Lin, Li Luo, Qi Wei, Runsheng Wang, Cheng Zhuo, Xunzhao Yin, Fei Qiao, and Huazhong Yang. 2021. 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* 69, 1 (2021), 232–243.
- [71] Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. 2015. A Dynamic Vision Sensor With 1% Temporal Contrast Sensitivity and In-Pixel Asynchronous Delta Modulator for Event Encoding. *IEEE Journal of Solid-State Circuits* 50, 9 (2015), 2149–2160.
- [72] Christopher Young, Alex Omid-Zohoor, Pedram Lajevardi, and Boris Murmann. 2019. A data-compressive 1.5/2.75-bit log-gradient QVGA image sensor with multi-scale readout for always-on object detection. *IEEE Journal of Solid-State Circuits* 54, 11 (2019), 2932–2946.
- [73] Ying-Ju Yu and Carole-Jean Wu. 2018. Designing a temperature model to understand the thermal challenges of portable computing platforms. In *2018 17th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*. IEEE, 992–999.
- [74] Yuhao Zhu, Anand Samajdar, Matthew Mattina, and Paul Whatmough. 2018. Euphrates: Algorithm-SoC Co-Design for Low-Power Mobile Continuous Vision. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, 547–560.