# Swordfish: A Framework for Evaluating Deep Neural Network-based Basecalling using Computation-In-Memory with Non-Ideal Memristors

Taha Shahroodi[1] Gagandeep Singh[2,3] Mahdi Zahedi[1] Haiyu Mao[3] Joel Lindegger[3] Can Firtina[3]
Stephan Wong[1] Onur Mutlu[3] Said Hamdioui[1]

[1]TU Delft [2]AMD Research [3]ETH Zürich

## Abstract

*Basecalling*, an essential step in many genome analysis studies, relies on large Deep Neural Networks (DNNs) to achieve high accuracy. Unfortunately, these DNNs are computationally slow and inefficient, leading to considerable delays and resource constraints in the sequence analysis process. A Computation-In-Memory (CIM) architecture using memristors can significantly accelerate the performance of DNNs. However, inherent device non-idealities and architectural limitations of such designs can greatly degrade the basecalling accuracy, which is critical for accurate genome analysis. To facilitate the adoption of memristor-based CIM designs for basecalling, it is important to (1) conduct a comprehensive analysis of potential CIM architectures and (2) develop effective strategies for mitigating the possible adverse effects of inherent device non-idealities and architectural limitations.

This paper proposes Swordfish, a novel hardware/software co-design framework that can effectively address the two aforementioned issues. Swordfish incorporates seven circuit and device restrictions or non-idealities from characterized real memristor-based chips. Swordfish leverages various hardware/software co-design solutions to mitigate the basecalling accuracy loss due to such non-idealities. To demonstrate the effectiveness of Swordfish, we take Bonito, the state-of-the-art (i.e., accurate and fast), open-source basecaller as a case study. Our experimental results using Swordfish show that a CIM architecture can realistically accelerate Bonito for a wide range of real datasets by an average of 25.7×, with an accuracy loss of 6.01%.

## 1 Introduction

*Basecalling* is the first computational step required to translate noisy electrical signals generated by modern sequencing machines to strings of DNA nucleotide bases (i.e., {A, C, G, T}), also known as DNA reads or simply reads [6, 12, 53, 60, 98, 107, 127, 131, 133]. The accuracy of basecalling directly affects the overall accuracy and the computational effort (in terms of required algorithms and their complexity and runtimes) of subsequent genome analysis steps. The speed of basecalling also determines how fast one can run through all computational steps of a genomic study [107, 120, 134]. Therefore, accurate and fast basecalling is critical for advancing genomic studies that hold the key to unlocking the potential of precision medicine, facilitating virus surveillance, and driving advancements in healthcare and science [5–7, 13–15, 28, 29, 34, 41, 42, 62, 67, 84, 87, 103, 137, 142].

Current state-of-the-art (SotA) basecallers leverage Deep Neural Networks (DNNs) to achieve high accuracy [31, 96, 105, 120, 140, 149]. However, SotA DNN-based basecallers encounter different shortcomings when implemented using different approaches. Specifically, DNN-based basecaller designs on Central Processing Units (CPUs) and Graphics Processing Units (GPUs) face multiple major shortcomings: (1) they are computationally intensive and slow [107, 120, 134], (2) they require extensive data movement between the processor and memory [16, 17, 79], and (3) they are limited by the use of costly hardware, such as expensive SRAM memories that require 6 transistors for storing only 1 bit of information [30, 102]. When implemented on a hardware accelerator, these DNN-based basecallers face two other limitations: (1) They rely on costly floating-point (FP) computations, which place high demands on the required system's memory bandwidth and compute units with FP capability. This makes hardware acceleration difficult due to the large number and size of neural network model parameters. (2) They use costly Machine Learning (ML) techniques such as skip connections[1] [96, 123, 140], leading to added computation, memory, and storage overheads (e.g., to store the activation parameters that are fed to the last layers of the NN) [120]. Therefore, over the past decade, both industry and academia [27, 68, 101, 111, 115, 119] have explored the use of Computation-In-Memory (CIM)[2] using memristor-based devices to accelerate DNNs.

This growing interest in using CIM for resolving the shortcomings of DNNs is driven by two main factors: (1) the potential of the CIM paradigm to process data where it resides to reduce the large performance and energy overheads of data movement and (2) the analog operational properties of these nanoscale emerging technologies (e.g., memristors) that intrinsically support efficient Vector-Matrix-Multiplication (VMM), multiple of which are used to implement a Matrix-Matrix-Multiplication (MMM) that is the most dominant operation in DNNs. However, the memristor-based CIM solutions for basecalling can greatly degrade the DNN inference accuracy due to (1) the limited quantization levels supported by memristor devices [27, 111] and (2) non-idealities of memristive devices and circuits used to adopt memristor-based memory arrays, such as sneak paths [48, 118] and the non-linearity of peripheral circuitry [58, 83, 147]. To propose viable solutions for accelerating the large-scale DNN-based basecallers, these aspects must be considered at all computing stack layers, i.e., application, architecture, and device. Such considerations are only possible with a framework capable of evaluating the impact of the non-idealities in memristor-based CIM architecture on the end-to-end basecalling accuracy.

[1]Skip connection is an ML technique that allows skipping a few neural network layers and forwarding the output to the input of a layer further ahead.
[2]Interchangeably, also referred to as Processing-In-Memory (PIM) [85].

This framework should also be able to account for the overhead that the solutions to overcome the accuracy loss may bring.

To this end, we propose *Swordfish*, a modular and extensible hardware/software co-design framework that allows us to (1) evaluate the impact of memristor non-idealities and CIM limitations on the accuracy and performance of basecalling and (2) investigate potential mitigation techniques and measure their effect on accuracy for each non-ideality (**Contribution #1**). Swordfish is used to investigate the acceleration of basecalling via emerging computing paradigms and technologies. Specifically, with Swordfish, we comprehensively investigate the potential of accurate acceleration of a SotA basecaller (Bonito) on a SotA CIM architecture (PUMA [9]) by accounting for the non-idealities of the underlying devices and technologies of the underlying architecture, for the first time (**Contribution #2**). Swordfish integrates real-world applications with multiple critical comparison metrics, distinct mitigation strategies to tackle the challenges of novel hardware, and comprehensive real measurements to guide the modeling of memristors. Our evaluations using Swordfish show that on a wide range of real genome datasets, PUMA accelerates Bonito, a SotA basecaller, by an average of 25.7× realistically (i.e., the average throughput improvement is 25.7× when we consider essential mitigation techniques to prevent huge accuracy loss). This performance still comes at the cost of a 6.01% accuracy loss (Section 5). Our evaluations also yield several key suggestions and recommendations for DNN, hardware, and system designers of future emerging accelerators with memristors for DNN-based basecallers and other applications that have two most important metrics (e.g., accuracy and performance) to consider in their evaluation (**Contribution #3**). Specifically, our investigation using Swordfish results in multiple unique insights: (1) Our results challenge the prevalent assumption that DNN-based applications will automatically succeed on memristor-based CIM due to inherent redundancy in large neural networks, (2) combining mitigation techniques at only one abstraction level (e.g., circuit or system level) does not necessarily improve the accuracy loss as they can potentially go against each other, and (3) combining multiple mitigation techniques at the circuit and system levels can offset the accuracy loss induced by non-idealities significantly.

## 2 Background and Motivation

This section briefly discusses the necessary background and motivation for this work. We refer the reader to comprehensive reviews [6, 18, 45, 85, 98] for more details.

### 2.1 Genome Sequencing Pipeline

The genome sequencing pipeline consists of computational steps we employ to acquire genome sequences as strings of DNA characters (i.e., {A, C, G, T}) [6, 12, 53, 60, 98, 107, 127, 131, 133] for subsequent analysis in bioinformatics, e.g., cell type identification, identification of marker genes, and variant detection.

Although, currently, the most available data and tools in the genomics realm are for short reads [20, 39] (mainly produced by Illumina sequencers), working with highly accurate long genome sequences is generally favorable as they reduce the computational cost of reconstructing the genome. For this reason, there is a large momentum towards accurate long-read sequencing [6]. Our work focuses on finding solutions and analysis tools that target long reads while also not discarding tools (e.g., GenAx [39] and GenASM [20]), designed for short reads. A leading method for long-read sequencing is the nanopore sequencing technology. Nanopore sequencers [90, 93, 94] translate raw signal squiggles into bases (A, C, G, T) using complex neural networks. Today, Oxford Nanopore Technologies (ONT) is a company that produces the most commonly used sequencers based on Nanopore technology.

Fig. 1 illustrates the nanopore genome sequencing pipeline [107] and the placement and execution time breakdown of each of its steps. We use SotA tools for each step and run the tool on the datasets described in Section 4.

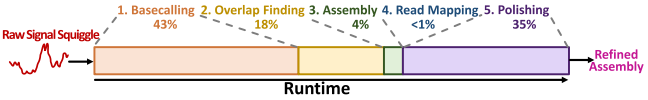


**Figure 1: Overview of the nanopore genome sequencing pipeline and execution time breakdown of different steps.**

We make two main observations. First, basecalling is the first computational step in the pipeline. Second, basecalling dominates the execution time of a single run in the pipeline. These steps make up more than 40% of the entire execution time. Our empirical observation aligns with those in prior works [33, 81, 107].

### 2.2 Basecalling

Basecalling is responsible for converting raw electrical signals produced by a nanopore sequencer to digital genome symbols, i.e., [A, C, G, T] [12, 53, 60, 127]. Recent works [92, 95, 96, 134] heavily investigate the use of DNNs for basecalling as they can provide high accuracy than Hidden Markov Model (HMM) based techniques [91].

There are generally two approaches for improving the accuracy and/or performance of a basecaller: 1) software-based and 2) hardware-based. Software-based methods propose new algorithms (e.g., DNNs [95, 96, 140] instead of HMMs [91]) or faster and/or smaller DNN architectures [120, 140]. Hardware-based approaches propose various hardware platforms for the target algorithm (i.e., DNN or HMM) to improve performance with (hopefully) small impact on accuracy [81, 120].

We observe four main shortcomings in SotA basecallers, which limit their execution time and/or hardware acceleration:

- SotA basecallers are slow and energy inefficient. For example, Guppy basecalls 3 Giga basepairs (Gbps) in ~6 hours while a following step in the genomics pipeline, such as read mapping using minimap2 [71] takes only ~0.11 hours [120].
- SotA basecallers use DNN models with costly skip connections [123]. For example, Bonito needs an additional ~21% of model parameters (along with associated memory and storage overheads) for skip connections and requires additional computation on them. Note that a skip connection permits bypassing certain layers within the neural network, transmitting the output of one layer as the input to subsequent layers [123]. These connections are costly because they (1) typically force the network to perform additional computation, for example, to match the channel sizes, (2) incur extra memory and storage overhead, as they require storing the activation parameters that are fed to the later layers [16, 17], and (3) incur additional off-chip data movement overhead when these networks are run on conventional processor-centric hardware platforms, like CPUs and GPUs.

- SotA basecallers exploit 32-bit floating point precision for their model parameters [96, 134, 140]. This effectively increases (1) the required bandwidth and processing units, e.g., with FP compute capability, and (2) inefficiency in the hardware realization of the underlying models.
- SotA basecallers incur expensive data movement between the computation units and the memory units [79, 81, 120].

We emphasize that 40% of execution time spent on basecalling (Section 2.1), the first and arguably most critical step in the pipeline, is significant and worth accelerating. Today's best basecallers often underperform on SotA systems, generating bottlenecks. A potentially 40% decrease in genome analysis runtime implies a proportional reduction in power and energy, which is critical considering the extensive data and computational demands of modern genome analysis systems. Therefore, optimizing basecalling contributes greatly to improving the efficiency and sustainability of the genomics pipeline.

## 2.3 Memristor-based CIM and Associated Non-Idealities

Resistive memories or memristive devices, such as ReRAM, PCM, and STT-MRAM [59, 69, 119, 132], have recently been introduced as suitable candidates for both storage and computation units that can efficiently perform vector-matrix multiplication [138] and logical bulk bit-wise operations [26, 73, 113, 114, 139], as they can follow Kirchhoff's law inherently [121]. Therefore, many recent works [9, 26, 27, 111, 112, 139, 143–145] exploit these devices in their CIM architectures. Memristor devices also enjoy non-volatility, high-density, and near-zero standby power [73, 119, 139].

A typical memristor-based memory crossbar capable of VMM and other logical operations is shown in Fig. 2 [9, 26, 27, 111, 139] alongside its possible non-idealities.

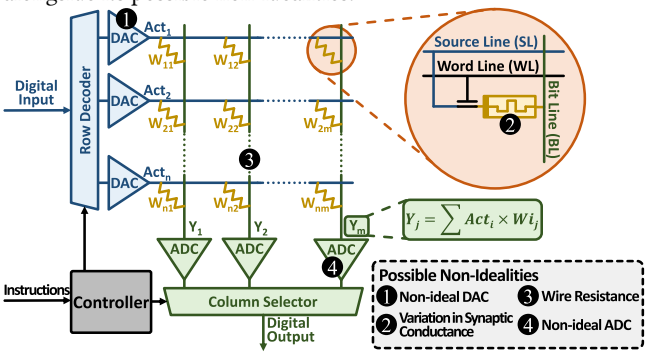


**Figure 2: Overview of memristor-based crossbar arrays and possible non-idealities.**

This memristor-based structure can suffer from at least four types of non-idealities or variations that can eventually affect the results of the enabled VMM operation, i.e., lead to errors in the VMM result: (1) The non-ideal digital to analog converter (DAC), due to the effective resistive load (known as $R_{Load}$) in its circuit [55], (2) Variation of synaptic conductance, which includes both imperfect programming operation (commonly known as write variations) and the process variation that exist in memristors [4, 23, 70, 148], (3) The wire resistance and sneak paths, due to imperfect wires (i.e., wires with different resistances) and the changes in the voltages of the internal nodes while performing a VMM operation [56, 148], and (4) non-ideal sensing circuit or analog to digital converters (ADCs), due to rigid or hard-to-accurately-change references used for distinguishing/sensing the end result [55, 144]. Our work focuses on these specific non-idealities inherent to memristor technologies in a CIM architecture. While we do not explicitly address other circuit challenges and non-idealities, we acknowledge their presence and the existing solutions developed to mitigate them in electronic systems. For example, crosstalk [104, 129, 130], which involves interference between adjacent circuit traces or wires, can indeed lead to data corruption and compromise information integrity. However, we focus on the specific non-idealities relevant to our hardware architecture, not crosstalk. Note that industry-standard techniques, such as shielding and layout design, decoupling components, ground and power distribution, signal timing and margins, ECC and scrubbing, isolation and shielding, and crosstalk-aware clock distribution, have been extensively studied and developed to mitigate crosstalk issues. We assume that similar techniques can be applied to address any potential crosstalk concerns in memristor-based CIM systems.

Recent works [9, 19, 27, 86, 111] report impressive performance and energy improvements for DNN models executed on memristor-based CIM architectures, mainly assuming idealized underlying hardware. Moreover, DNNs are known to be resilient to some noise [44, 46, 66, 125, 126, 128]. However, since memristor-based CIM architectures are indeed non-ideal and the resiliency of DNNs has a limit, to decide whether or not these platforms are indeed suitable for realizing our DNN-based basecaller, one needs to evaluate the impact of these non-idealities on the end-to-end application accuracy and account for the overhead that the solutions to overcome the accuracy loss may bring. Such a framework is missing among prior works and is a contribution of our work (Section 3).

## 2.4 Programmable Inference Architecture

PUMA (Programmable Ultra-efficient Memristor-based Accelerator) [9–11] is a complete set of (micro)architecture, simulator, and compiler that supports the execution of many ML applications, using memristor crossbars enhanced with general-purpose execution units. PUMA uses a spatial architecture and provides the necessary programmability and generality to execute a wide range of ML-based applications on memristor-based crossbars. For evaluations in Swordfish, we assume an PUMA-based architecture for two reasons. First, PUMA supports all the necessary types of NN layers in basecallers: CNN, LSTM, and linear. This is especially handy for our main target basecaller, Bonito. Second, the architecture, simulator, and compiler are open-sourced [10, 11] and well-documented for an extension, unlike many other rich architectures.

# 3 Swordfish Framework

Swordfish is a framework designed to guide the evaluation of CIM designs for DNN-based basecallers.

## 3.1 Swordfish Overview

Fig. 3 presents an overview of the Swordfish framework. Swordfish consists of 4 key modules:

- ❶ *Partition & Map* module that partitions and maps the Vector-Matrix-Multiplication (VMM) operations of the target DNN-based basecaller to the underlying CIM platform,

- ❷ *VMM Model Generator* module that generates an end-to-end model for possible non-idealities and errors of a VMM operation considering the underlying technology in the CIM design,
- ❸ *Accuracy Enhancer* module that implements online and offline mitigation techniques to counter accuracy loss, and
- ❹ *System Evaluator* module that analyzes the accuracy and throughput of basecaller while also providing an area overhead.

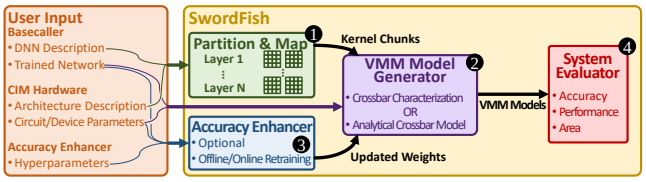


**Figure 3: Overview of Swordfish framework.**

We emphasize that the accuracy analysis in the System Evaluator module is critical and unlike evaluations of conventional platforms, e.g., Field-Programmable Gate Arrays (FPGAs) or GPUs. Its importance stems from the abundance of the underlying non-idealities, variations, limitations, and hardware perturbations of the emerging hardware paradigms [57]. From now on, we refer to the proposed framework as *Swordfish* and the actual implemented memristor-based CIM design for our target basecaller Bonito as *SwordfishAccel*.

## 3.2 Partition & Map

To run the DNN of a basecaller on a CIM architecture, one should map each of the VMM operations in the target DNN to the analog memory arrays and the rest of the operations to the digital peripheral circuitry. The Partition & Map module takes care of this task in Swordfish by dividing individual functions of the basecaller into the analog or digital components of the underlying architecture. This process is required one time for every basecaller and has two steps.

In the first step, Swordfish decides which memory crossbars will perform each VMM operation of each layer. For Bonito basecaller, Swordfish decides which memory crossbars handle the VMM of the first convolutional layer and which crossbars are responsible for the VMMs of the following LSTM and linear layers. Swordfish assumes that all the underlying crossbars have the same size and readout peripheral circuitry (e.g., ADCs).

In the second step, Swordfish decides how it maps the weights to each crossbar. Swordfish supports different programming/writing techniques for memristor devices, such as write-read-verify (WRV) and Set/Reset pulse programming.

In mapping and evaluation, Swordfish makes the following widely common design choices:

- The input streams into the first layer of DNN. Swordfish does not divide the input into chunks and leaves this task to the host. Doing so helps Swordfish to evaluate the maximum throughput of a basecaller [81, 96], independently of the input size.
- The next layer starts its computation as soon as the previous layer of the basecaller produces enough values. This is also a common assumption for evaluating the maximum possible throughput of a DNN in simulation [9, 111].
- Multiple crossbar arrays can be simultaneously active and perform the necessary operations (VMM and other operations necessary for the target DNN, such as activation. This assumption ensures that full chip utilization is not limited due to power constraints. One can consider this parallelism to be analogous to the concurrent activation of multiple subarrays in different banks and bank groups in traditional DRAM [21, 109, 110].
- Swordfish optimizes its design decisions for the highest achievable accuracy, throughput, and memory utilization in the stated order. This is a common priority order for optimizations in basecallers [81, 96, 97].

## 3.3 VMM Model Generator

VMM Model Generator is responsible for generating the non-ideal output per each VMM required by the basecaller. VMM Model Generator differentiates between constraints and non-idealities. This is essential in a CIM design where non-idealities or constraints do not necessarily lead to a loss in the accuracy of the application. To model the effect of these constraints and non-idealities on the accuracy of an application, Swordfish considers them at the lowest-level building block where they aggregate, i.e., where their results merge. In a memristor-based CIM architecture for a DNN-based basecaller, such an effective place to consider the effects of constraints and non-idealities is the VMM operation output. Therefore, the VMM Model Generator in Swordfish focuses on assessing the effects of each factor on a VMM operation, while our evaluations and analyses assess the end-to-end basecalling metric.

This module takes three types of inputs. First, it takes the results of the previous module (i.e., ❶ Partition & Map in Fig. 3) to determine the size of the VMM. Second, it takes the circuit and device description (i.e., constraints and non-idealities) that can affect accuracy. Examples inputs in this category are (1) the level of quantization, (2) the circuit variations (e.g., in inputs (e.g., DACs), wires, and outputs (e.g., ADCs) device), and (3) device variations. Third, it takes the weights of the target basecaller, which can be provided directly by the user or the Accuracy Enhancer module that applies multiple training mechanisms (Section 3.4). The module outputs the non-ideal output vector per each input vector and weight matrix (i.e., the expected vector result for a VMM).

Swordfish supports two different approaches for modeling a VMM. The first approach is to use a pre-calculated library of measurements on actual devices. The second approach is to use an analytical model (e.g., a fast crossbar model (FCM) [55]). Section 5 evaluates these approaches separately.

In the first approach, Swordfish queries a library that, for a given array size and input vector, returns an output vector randomly chosen from many ($\geq 10^4$) possible outputs based on measurements on an actual crossbar with the same dimensions as the length of the active input vector. The measurements in the library already contain all the possible non-idealities in the target VMM operation, i.e., non-idealities that may arise from DACs, ADCs, circuits, and devices in the crossbar. One can build this library by measuring multiple tiles several times. For each of these measurements, one should program the initial values of memristors within a tile with the weight values of the target DNN to be evaluated on Swordfish. In this paper, the distinct initial resistance states are based on the Bonito basecaller [96]. The random choice from the library aims to account for variations and non-idealities among different memristor-based tiles, which can arise from different initial values of each memristor device and/or manufacturing differences. By integrating real

measurements and accounting for tile-to-tile differences, we believe our methods accurately reflect non-ideality distribution in practical settings. Although this approach accurately represents the VMM operation considering many possible non-idealities, it lacks the flexibility of separately studying or measuring the effects of each possible error due to different non-idealities. This approach is also limited to the crossbar configurations (for example, crossbars of 64×64 and 256×256) to whose measurements one has access (Section 4).

In the second approach, Swordfish utilizes existing analytical models that are available for ADCs, DACs, and variation profiles of the underlying devices in the crossbar. Fig. 4 illustrates the steps Swordfish uses in its VMM Model Generator for this approach.


**Figure 4: An overview of the VMM Model Generator's second approach: using analytical models.**

In Fig. 4, Swordfish applies the analytical model for a non-ideal DAC model (❶) to the input vector of the VMM operation (❶) and obtains the non-ideal input voltages as the output vector (❷). Swordfish then applies this new vector to a crossbar with an updated non-ideal weight matrix (❷), where non-idealities have been applied to the original weight matrix (from the VMM operation) based on the expected variations of each cell, which are usually obtained based on generic characterization of memristor-based crossbar arrays, i.e., without any peripheral circuitry or target weights specific to a particular DNN. The output is considered a non-ideal output current (❸) that Swordfish applies to a model of non-ideal ADC (❸) and obtains the output vector (❹), an output vector that might contain some errors.

Fig. 5 presents an overview of how Swordfish models the crossbar non-idealities for the second approach (i.e., the analytical model in the VMM Model Generator module) (❷ in Fig. 4). For this, Swordfish first takes the crossbar instances (① in Fig. 5) from the Partition & Map module. Swordfish considers these crossbar instances as separate matrices with digital weights (②). Then, Swordfish uses a non-linear model for the synaptic device states (③) to map the weight matrices of digital weights into ideal corresponding conductance matrices (④). After that, Swordfish applies to these metrics the synaptic variations for the crossbar (⑤) that are determined from an analytical model based on the estimated behavior of memristor devices within a crossbar array. The output consists of the same number of matrices, but now with adjusted weights (⑥). Swordfish finally applies to those matrices the profile of all known circuit-level non-idealities (⑦) by adding representative metrics for these non-idealities. The output consists of matrices accounting for all variations and non-idealities (⑧).

## 3.4 Accuracy Enhancer

Since accuracy is a critical metric in basecalling, Swordfish applies several mitigation techniques to deal with the non-idealities and their induced errors on the VMM and/or basecalling. More specifically, Swordfish supports four different accuracy enhancement techniques: (1) analytical variation-aware training (VAT) (offline), (2) knowledge distillation (KD) training, (3) read–verify–write (R-V-W) training, and (4) random sparse adaptation (RSA) retraining (online).

*3.4.1 Analytical Variation-Aware Offline Training.* Swordfish supports variation-aware training (VAT) [24, 63, 78, 80] during the training of a target DNN as the simplest method to enhance the accuracy loss due to (1) quantization and (2) possible resistance variations per weight, which can be analytically or experimentally measured. Existing works randomly inject faults into the weights of the DNN [38], or model the potential errors at the end of each layer [38, 80]. Similarly, Swordfish utilizes the crossbar characterization for the errors per VMM (i.e., the error library in the first approach in VMM Model Generator) or an analytical crossbar model for the errors per VMM (i.e., as in the second approach in VMM Model Generator). Swordfish injects the modeled errors in the training and considers the rest of the devices unaltered. Swordfish repeats this process for each VMM and every layer and then retrains the basecaller network. This way, Swordfish ensures that its retraining yields a better estimate for the errors arising from non-idealities in the crossbar.

*3.4.2 Knowledge Distillation-based Variation-Aware Training.* In addition to offline VAT based on injecting random errors or potential errors per layer discussed in Section 3.4.1, Swordfish is capable of supporting the knowledge distillation (KD) approach as a VAT as well, i.e., Swordfish exploits knowledge/weights that exist in an ideal (typically a FP32-based) basecaller baseline to guide the training of SwordfishAccel, our memristor-based CIM design for Bonito. In KD, two models exist: (1) the teacher (an ideal implementation using high precision data format, e.g., FP32-bit format) and (2) the student (SwordfishAccel quantized to 16-bit-width fixed-point presentation for both weights and activations). The goal is to mimic the teacher's output in the student by minimizing a loss function where the target is the result of applying the softmax on the quantile function associated with the standard logistic distribution (i.e., logit) of the teacher's training [47]. We refer the reader to previous works on KD [22, 47] for further detail on how a loss function can be implemented in such a system to minimize the difference of SwordfishAccel's output and the teacher model's softmax output.

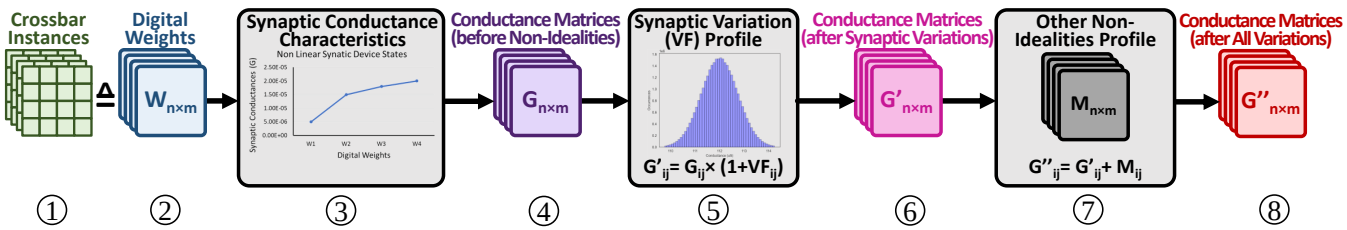


**Figure 5: An overview of modeling crossbar non-idealities in Swordfish.**

*3.4.3 Read-Verify-Write (R-V-W) Training.* Read-Verify-Write (R-V-W) is a conventional error mitigation technique for non-ideal memristor-based memories that provides cell-by-cell error compensation. R-V-W is used in open-loop-off-device (OLD) [77] where R-V-W programming and sensing loop help the actual resistance of the device to converge to the expected target resistance. This method involves many read-and-write operations and feedback control for memristors, making R-V-W a slow technique to mitigate accuracy loss. Note that to improve the accuracy in R-V-W, we need to increase the fraction of the retrained weights (memristor devices in our case), increasing the cost of the mitigation technique.

*3.4.4 Random Sparse Adaptation Online Retraining.* Swordfish uses random sparse adaptation (RSA) [22] to map the learned DNN model to SwordfishAccel. RSA is used to mitigate the performance overhead of R-V-W technique [49, 77]. RSA by itself prevents only some of the non-idealities from being materialized as inaccuracies and can be an offline mechanism. However, SwordfishAccel combines it with an online training mechanism.

For its online retraining using RSA, Swordfish places a small on-chip SRAM-based memory next to memristor-based crossbars and distributes the learned DNN model (i.e., weights) between this SRAM and memristor-based crossbars. The key idea Swordfish uses is to map the weights that otherwise would map to error-prone memristor devices to reliable SRAM cells. If one has access to the exact profile of the underlying memristor-based memory crossbars, one can exploit the knowledge on which memristors and columns are more error-prone and use this knowledge to decide which weight to map into the crossbar and which one to the SRAM. In our evaluations of Swordfish, we use this knowledge whenever we use the chip measurements already used in the first approach of the VMM Model Generator. However, Swordfish can also randomly choose memristor devices in the crossbar and map (i.e., hardwire) them to the SRAM. Random choice is the next best option without knowledge about the exact error pattern of a memristor-based crossbar. We used this method whenever we used the second approach (i.e., analytical model) in the VMM Model Generator (Section 3.3).

Fig. 6 presents how SwordfishAccel adopts RSA with an online retraining mechanism (e.g., KD) in a three-step approach:

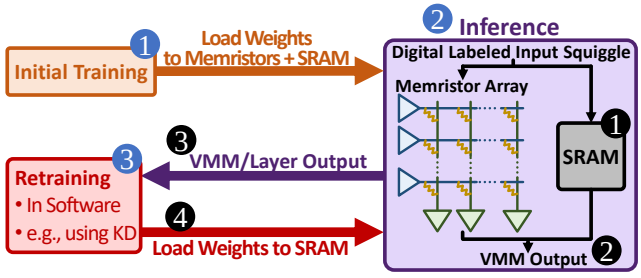


**Figure 6: Swordfish's online error mitigation via RSA.**

(1) In the first step (❶), SwordfishAccel trains the original Bonito and loads the initial weights from the Bonito DNN model into the assigned memristor crossbar and the SRAM (❶). SwordfishAccel considers this model as the initial model for the student in KD.

(2) In the second step (❷), SwordfishAccel performs a VMM operation as usual. However, whenever one or more of the assigned weights to SRAM (i.e., error-prone memristors or randomly chosen ones in Swordfish) is involved, SwordfishAccel reads the value from the SRAM memory instead of the memristor device. Swordfish does this by passing the inputs of corresponding devices through the SRAM value instead of the crossbar, zeroing the input for that particular memristor in the crossbar, and then summing up the values of both paths (❷).

(3) In the third step (❸), SwordfishAccel returns the results of the VMM operation of each crossbar (❸) to the retraining component (KD in our example in Fig. 6) and performs online training on only the weights that are mapped to the SRAM memory to improve the accuracy loss due to non-idealities. Note that SwordfishAccel considers the non-ideality models of crossbars, ADCs, and DACs to the student model for every training batch and trains the student. This includes both the initial training in Step ❶ and retraining in Step ❸.

(4) SwordfishAccel then loads the new weights to the SRAM near the crossbars (❹) and repeats Steps ❷ and ❸.

SwordfishAccel uses KD-based variation aware training for its Step ❸ in Fig. 6 online retraining. However, any other retraining method can also replace KD in our example. Note that all the parameters are already quantized to 16-bit fixed-point precision to present the model in SwordfishAccel accurately. Swordfish leverages the weights from the converged teacher model to improve the convergence of the student model.

RSA in Swordfish comes at the price of extra area overhead for the considered on-chip SRAM memory, storage in the memory controller for mapping metadata, summation of the output from the crossbar with on-chip memory, and some additional control logic evaluated in Section 5.

## 3.5 System Evaluator

The System Evaluator module puts the results of all previous modules of Swordfish together to evaluate the target DNN.

As inputs, this module takes the execution time for each VMM operation, the accuracy of each VMM operation for the last layer of the DNN (as it determines the final accuracy of the DNN), the number of active crossbars in each step of Swordfish, and information in peripheral circuitry.

The System Evaluator module has 3 outputs:

(1) **Accuracy:** The System Evaluator module outputs an accuracy number for the evaluated DNN. In SwordfishAccel, this number shows the accuracy of the basecaller, commonly known as *read accuracy*, which is the fraction of the total number of exactly matching bases of a read to a reference to the length of their alignment (including insertions and deletions).

(2) **Basecalling throughput:** The System Evaluator module outputs a number for inference throughput of the target DNN. In SwordfishAccel, this number is the basecalling throughput, defined as kilo-basepairs generated by the basecaller per second ($\frac{Kbp}{s}$). The higher the basecalling throughput, the better. This is the most important metric to evaluate a basecalling accelerator's performance. Our throughput evaluations in SwordfishAccel include the time required for read and write time for the inputs and outputs, respectively.[3]

(3) **Area overhead.** The System Evaluator module of Swordfish also reports area overhead based on the underlying architecture

[3]We use this command line in Linux: /usr/bin/time -v.

to account for the overheads of a dedicated accelerator, e.g., SwordfishAccel.

### 3.6 Swordfish Evaluation Challenges

Comprehensive, fair, and practical evaluation of Swordfish is challenging for two main reasons. First, most of the SotA basecallers are either not open-source [50, 81, 134] or support only specific reads [97]. Second, current simulators and frameworks mimicking memristor-based CIM designs are either not open-source, do not consider the underlying non-idealities of the devices, or only support a very limited number of non-idealities, emerging technologies, or neural networks [55, 76].

To evaluate Swordfish despite these challenges, we take two representative examples. Specifically, for the first challenge, we primarily compare our method with Bonito [96], an open-sourced, universally applicable tool currently under active development and maintenance by ONT (Section 2.1). Bonito stands out for its exceptional accuracy and performance over its predecessors like Guppy [134] and does not face the limited support for reads (e.g., Dorado [97]) or lack of open-source implementation and training code (e.g., Helix [81], Halcyon [65], Guppy [134], and SACall [50]). For the second challenge, we consider PUMA architecture as the baseline architecture for the two reasons mentioned in Section 2.4.

## 4 Evaluation Methodology

### 4.1 Implementations and Models

For the performance and area studies, we significantly extended the PUMA simulator and PUMA compiler to account for (1) Bonito's DNN architecture, (2) updated configurations in Core Architecture of PUMA [9] based on our memory models and the TSMC 40 nm [61] technology node used for peripheries, and (3) performance and area overheads introduced by non-idealities of memristors and their mitigation techniques. Note that we use Synopsys Design Compiler [122] and synthesize the additional components of our design in the target technology to obtain their execution time, power, and area. We apply the prominent technology scaling rules [106] to the configuration numbers of the PUMA architecture to ensure all of our design components are based on the same technology node.

For accuracy analysis (in both training and inference phases), we also extensively modified Bonito's open-source implementation [96] to consider the device characteristics and limitations of the architecture. Unfortunately, PUMA does not allow us for such analysis as it considers the effects of only quantization and write variations on accuracy.

We utilize prototyped cross-array memristors as our memory arrays and capture the variations in their spatiotemporal conductivity, execution time, and area overhead of necessary operations. We project our characterization results of real memories to our DNN evaluations. We also build a statistical model from our measurements to capture the full picture of a larger memory model for large-scale variations, timing, and area parameters. This model contains four types of variations: (1) input DACs, (2) synaptic variations, (3) wire resistance, and (4) output ADCs. The memory prototypes and models used for evaluations and simulations are based on the results of the EU project MNEMOSENE [82], concluded in 2020, generously provided by the involved parties. The results have been tested heavily during the project and by various metrics found in the related literature. Table 1 shows the main parameters of our memristor-based crossbars.

| | |
|---|---|
| **Technology and device** | ReRAM $HfO_2/TiO_x$ [61] |
| **Cell configuration** | 1T1R (NMOS T: 460 nm/40 nm |
| **HRS/LRS** | 1 MΩ/10 kΩ |
| $\mathbf{n}_{min}/\mathbf{n}_{max}$ | 0.03, 30 |
| **Array Sizes** | 64×64 and 256×256 |
| **SA** $\mathbf{V}_{min}$ | 40 mV |

**Table 1: Our array and device configurations.**

Our study specifically evaluates Swordfish on ReRAM memristors for three reasons. First, the availability of actual chip measurements is essential for our non-ideality-centered study. Second, lower energy costs for writing/programming than alternatives like PCM. Third, ReRAM's established status within the memristor family provides reliable baselines and intuitions for device-level features, enhancing the credibility of our proposal.

### 4.2 Simulation Infrastructure

We ran our baseline Bonito basecaller and software implementation of Swordfish on a 128-core server with AMD EPYC 7742 CPUs [8], 500GB of DDR4 DRAM, and 8 NVIDIA V100 [88] cards. We train and evaluate Swordfish accuracy and software results on our NVIDIA cards (with 32-bit floating-point precision). We use the nvprof profiler [141] for the profiling experiments on GPU.

### 4.3 Evaluation Metrics

We use metrics output by the System Evaluator module for our comparisons. Section 3.5 clarifies these metrics.

### 4.4 Datasets and Workloads

Table 2 provides datasets from a MinION R9.4.1 flowcell [135, 136] we use in our evaluations.

| | Dataset (Organism) | # Reads | Reference Genome Size (bp) |
|---|---|---|---|
| D1 | Acinetobacter pittii 16-377-0801 | 4,467 | 3,814,719 |
| D2 | Haemophilus haemolyticus M1C132_1 | 8,669 | 2,042,591 |
| D3 | Klebsiella pneumoniae NUH29 | 11,047 | 5,134,281 |
| D4 | Klebsiella pneumoniae INF042 | 11,278 | 5,337,491 |

**Table 2: Read and Reference Datasets for our Basecalling Evaluation.**

## 5 Swordfish Evaluation

We first use Swordfish to investigate the impact of constraints and non-idealities of a PUMA-based architecture (Section 2.3) on the accuracy of the Bonito basecaller [96]. We call this design the Ideal-SwordfishAccel, as it achieves the highest performance for our memristor-based hardware accelerator without any accuracy enhancement technique. We then explore the effect of the accuracy enhancement mechanisms in Swordfish applied to deal with the inaccuracies of the memristor-based accelerator as it affects the Bonito basecaller's accuracy. The results of this design are presented under Realistic-SwordfishAccel.

### 5.1 Effect of Quantization on Accuracy without Accuracy Enhancement

Since both the weights and activations in the original DNN are in FP32 format, Swordfish can opt for quantizing one or both of them. The degree of the quantization can differ depending on how much

each parameter impacts the overall accuracy. Swordfish considers seven different configurations: the default configuration (DFP 32-32), where weights and activations use the FP32[4] format, and 6 FPP X-Y[5] formats, where X and Y denote the fixed-point precision of weights and activations, respectively. Swordfish currently only supports power-of-two precision levels for its quantized configurations. Table 3 presents the accuracy of different configurations.

| | DFP 32-32 | FPP 16-16 | FPP 8-8 | FPP 8-4 | FPP 4-8 | FPP 4-4 | FPP 4-2 |
|---|---|---|---|---|---|---|---|
| D1 | 97.32% | 97.32% | 97.12% | 97.12% | 95.42% | 95.62% | 93.62% |
| D2 | 97.32% | 97.32% | 96.72% | 96.72% | 94.92% | 95.42% | 92.42% |
| D3 | 97.32% | 97.32% | 96.02% | 95.82% | 93.62% | 95.12% | 93.72% |
| D4 | 97.32% | 97.32% | 96.42% | 96.42% | 94.22% | 95.32% | 93.62% |

**Table 3: Accuracy evaluation after quantization.**

We make two major observations. First, Bonito's architecture can tolerate some quantization level without accuracy loss. More specifically, across all evaluated datasets, quantization down to 16 bits does not affect the accuracy at all, and quantization down to 8 bits reduces the accuracy by less than 9% even in extreme cases. We conclude that Ideal-SwordfishAccel can still reduce the precision of its network from a 32-bit FP format to 16-bit-width fixed point precision without accuracy loss. This way, Ideal-SwordfishAccel can (1) accelerate the network on a platform limited to fixed point format representation and (2) improve the energy efficiency of the network via lower data precision. This observation is on par with similar studies [54, 111, 120] exploiting quantization as a technique to improve the performance and energy efficiency of a DNN with a negligible accuracy loss.

Second, tolerance to quantization varies depending on the input dataset. This makes the effect of quantization on accuracy workload-dependent. However, the accuracy drop for different quantization configurations follows more-or-less a similar trend irrespective of the dataset, i.e., they all follow a decreasing trend with reduced data representation. We conclude that Swordfish's understudy network (Bonito) tolerates some quantization but will offer very low accuracy for extreme quantization (i.e., lower than 4-bit precision) irrespective of the dataset. We note that an accuracy drop of ~5% and higher is considered unacceptable for a future basecaller, as accuracy is the most critical metric in SotA basecallers. This observation is consistent with prior works on smaller [55] or different types of networks [120].

We conclude that quantization is a viable solution to tackle data representation constraints in hardware accelerators and, therefore, can be used in a framework such as Swordfish. However, accuracy loss due to quantization (applied with the expectance of accuracy loss due to variations and non-idealities) leads us to consider only down to 16 (or possibly 8) bits of precision for both weights and activations before a significant accuracy drop occurs. Therefore, the following studies consider only a 16-bit integer as the quantization level.

## 5.2 Effect of Non-idealities on Accuracy without Accuracy Enhancement

We examine the effect of four non-idealities on basecalling accuracy. The results presented in this section belong to the second approach of modeling non-idealities in the VMM Model Generator module, i.e., using analytical modeling (see Section 3.3).

[4]FP stands for floating point.
[5]FPP stands for fixed point precision.

*5.2.1 Effect of Write Variation on Accuracy.* Write variation can single-handedly impact the accuracy results of a VMM operation [22, 54]. Therefore, we analyze it separately.

Fig. 7 presents the effects of write variations on accuracy. The x-axis sweeps the write variation rate. The error bars account for the accuracy variations on different write variation rates over 1000 runs of the model. Since the models for write variation are circuit-dependent and have varying probabilities of affecting the stored/programmed data, this methodology provides us with a better insight into the effect of this non-ideality on accuracy.

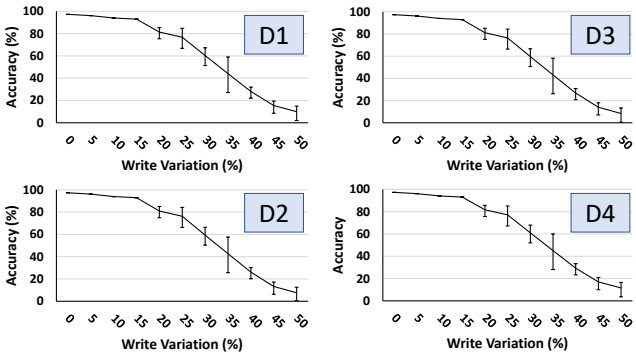


**Figure 7: Accuracy after taking into account write variation.**

We make two main observations. First, slight write variation can lead to a significant drop in the accuracy of end-to-end basecalling. To a great extent, this is on par with previous works' observation of the write variation impact on VMM accuracy [22, 54]. For example, the accuracy drops vary from 3.30% to 87.34% for D1 and from 3.24% to 85.76% for D4.

Second, the exact accuracy loss depends on the input dataset, i.e., the accuracy is workload-dependent and varies for the same write variation among different subfigures in Fig. 7. For example, for the same write variation rate of 25%, the accuracy on our two datasets (i.e., D2 and D4) can vary by 0.93%.

We conclude that write variation in Ideal-SwordfishAccel can debilitate the basecalling process significantly. In other words, write variation can eliminate all the potential performance and energy efficiency benefits of such a memristor-based design if not mitigated correctly. Therefore, unlike the quantization constraint, we should closely control the write variations in any future design for an acceptable basecaller. Fortunately, some previous works [22, 37, 100] propose mitigation techniques that, when combined, can provide us with reasonable (e.g., amount of $\leq$ 10%) write variation. From now on, we consider only up to 10% write variation (as defined in Section 2.3) in our evaluations.

*5.2.2 Effect of Combined Non-idealities on Accuracy.* Fig. 8 and Fig. 9 show the accuracy after considering all other sources of non-idealities (see Section 2.3) for our four datasets on two different crossbar sizes of 64×64 and 256×256, respectively. The error bars show the distribution when considering 10% write variation over 1000 runs. For each dataset, Fig. 8 and Fig. 9 present the accuracy results for five configurations presented as individual bars in the figures. The first three bars from the left present the results for individual non-idealities, i.e., synaptic+wire resistances (*Synaptic+Wires*), sensing+ADC circuitry (*Sense+ADC*), and DAC+driver circuitry (*DAC+Driver*), respectively, that Swordfish accounts for in

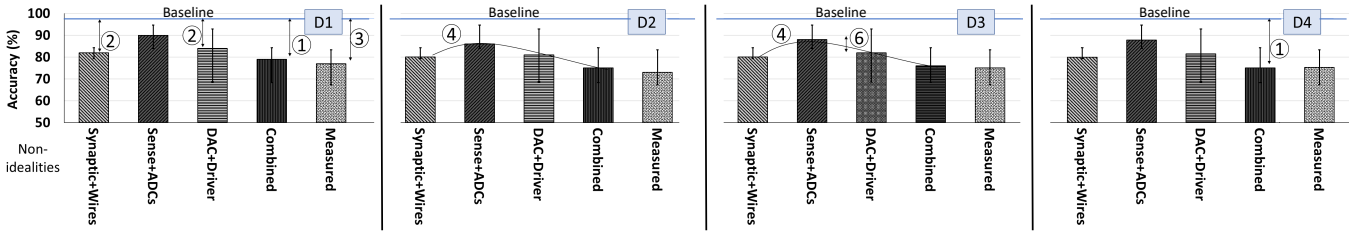


**Figure 8: Accuracy after taking into account non-idealities on 64×64 crossbars for the 4 datasets.**

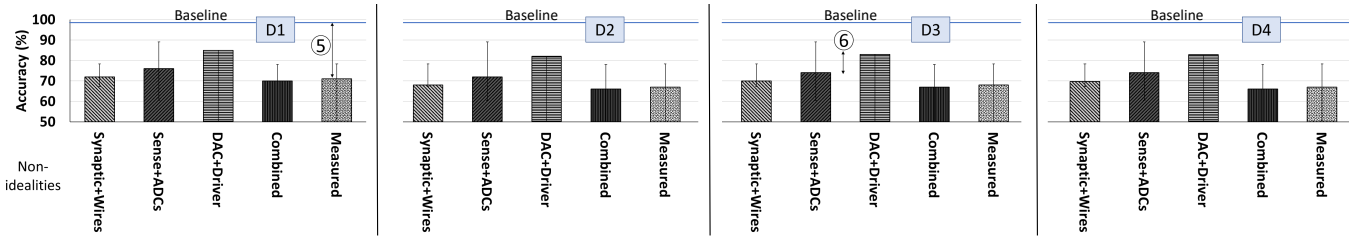


**Figure 9: Accuracy after taking into account non-idealities on 256×256 crossbars for the 4 datasets.**

its second approach of modeling non-idealities in the VMM Model Generator module, i.e., using analytical modeling (Section 3.3). The fourth bar, *Combined*, accounts for all the non-idealities from the same analytical model simultaneously. The fifth and last bar, *Measured*, considers all the non-idealities from the library of real chip measurements in the first approach of modeling non-idealities in the VMM Model Generator (see Section 3.3).[6] We make six main observations.

(1) A combination of non-idealities (i.e., each of the bars labeled with "Combined" or "Measured" or the 4th and the 5th bar per dataset in Fig. 8 and Fig. 9) leads to a significant accuracy loss irrespective of the dataset or crossbar size. For example, observe the accuracy loss when considering all the non-idealities in an analytical way (bars labeled as "Combined"). The accuracy loss varies from 18.32% to 31.32% (① in Fig. 8) across different datasets (i.e., D1 to D4). The same trend can be observed in Fig. 9.

(2) The impact of individual non-idealities (i.e., *Synaptic+Wires*, *Sense+ADC*, or *DAC+Driver*) on the accuracy (loss) is different. For example, observe the accuracy loss of *DAC+Driver* versus *Synaptic+Wires* in D1 (② in Fig. 8). For the same dataset, the accuracy loss varies from 13.32% for *DAC+Driver* to 15.34% for *Synaptic+Wires*. A similar difference also exists in crossbars of size 256×256 in Fig. 9.

(3) The accuracy loss for combined non-idealities is non-additive. For example, in D1, the total accuracy loss of *Measured* is 35.96% (③ in Fig. 8) yet the simple addition of numerical accuracy loss of *Synaptic+Wires*, *Sense+ADC*, and *DAC+Driver* totals 20.32%. We conclude that certain errors mask others.

(4) Accuracy loss values follow a similar trend irrespective of the dataset. See the trendlines ④ in Fig. 8 for D2 and D3. However, absolute accuracy loss values vary from one dataset to another.

(5) The smaller the crossbar, the lower the accuracy loss. For example, for D1, we have lower accuracy loss (of 20.32% versus 26.33%) when using a 64×64 crossbar compared to a 256×256 crossbar (③ in Fig. 8 vs. ⑤ in Fig. 9 for the *Measured* configuration). This is because a smaller crossbar has mostly smaller accumulative noise induced in wires of a smaller array.

(6) Different non-idealities affect the same dataset differently for different crossbar sizes. For example, the accuracy loss due to non-idealities in *DAC+Driver* is more dominant than those in *Sense+ADC* on a 64×64 crossbar, while this is the opposite for a 256×256 crossbar. See ⑥ in Fig. 8 and Fig. 9.

Even for small yet practical crossbars of size 64×64, the accuracy loss observed in this section under both *Combined* and *Measured* configurations in Fig. 8 and Fig. 9 is still significant (e.g., from 22.19% to 24.32%) and unacceptable for a basecalling step that affects many other steps of a genome sequencing pipeline. We conclude that non-idealities in the memristor-based CIM designs, especially when combined, can be detrimental to basecalling accuracy and must be accounted for and mitigated before considering such a design useful in any other aspect.

## 5.3 Effect of Accuracy Enhancement on Quantized Basecallers

Fig. 10 shows the results of applying Swordfish's accuracy enhancement techniques to a quantized Bonito basecaller. The x-axis presents six configurations for quantization as defined in Section 5.1. For each quantization configuration, we evaluate five accuracy enhancement techniques, namely *VAT*, *KD*, *R-V-W*, *RSA+KD* (see Section 3.4), and a combination of all techniques labeled as *All*. The y-axis shows the accuracy of each technique for the corresponding quantization configuration. The horizontal line marked as Baseline (DFP 32-32) is the baseline accuracy as defined in Section 5.1.

We observe that retraining with quantization is an effective way to mitigate the accuracy loss induced by quantization. Our results show that with only 150 extra retraining epochs, accuracy improves by 5% on average, for a basecaller quantized down to 8-bit. By applying all quantization-aware retraining methods that we discuss in Section 5.1, Swordfish can retain the same accuracy as the Bonito basecaller with 32-bit floating point precision. This result is in agreement with the prior work on different types of

[6]We leave the exploration of every possible combination of individual non-idealities to future work.

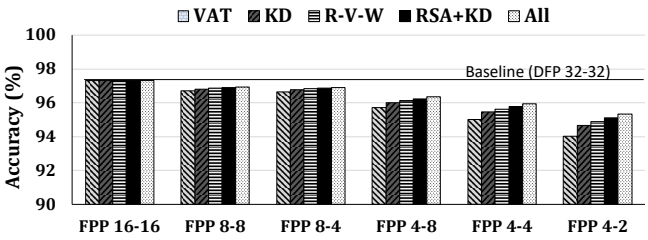


**Figure 10: Accuracy enhancement after quantization.**

neural networks [55]. However, Swordfish is the first work to show this result for genomic basecalling. From now on, we use 16-bit precision quantization for all evaluations we show in the remainder of this paper. We conclude that the proposed mitigation mechanisms effectively mitigate the accuracy loss due to a reasonable amount of quantization, e.g., from 32-bit to 16-bit in the Bonito basecaller.

## 5.4 Effect of Accuracy Enhancement on Non-idealities

*5.4.1 Effect of Accuracy Enhancement on Write Variation.* Fig. 11 presents the effects of our accuracy enhancement techniques (see Section 3.4) considering different write variation rates across our four datasets (D1-D4). The horizontal dotted line shows the baseline accuracy using DFP 32-32 (see Section 5.1) for the Bonito basecaller in all figures in Fig. 11. Fig. 11-(a)-(d) evaluate the effect of *VAT*, *KD*, *R-V-W*, *RSA+KD* separately. Fig. 11-(e) considers all of our accuracy enhancement mechanisms together (*Combined*), and Fig. 11-(f) averages the results of each accuracy enhancement technique over all the datasets (*Averaged*).[7] We make four major observations from Fig. 11.

First, individual accuracy enhancement mechanisms evaluated in Fig. 11-(a)-(d) all improve the accuracy. However, their effectiveness reduces as the write variation rate increases.

Second, the online mechanism (*RSA+KD*) in Fig. 11-(d) outperforms all the offline techniques in Fig. 11-(a)-(c). *R-V-W* in Fig. 11-(c) comes second in terms of accuracy. However, the difference between *RSA+KD* and *R-V-W* widens as the write variation rate increases.

Third, combining all the accuracy enhancement mechanisms (*Combined* in Fig. 11-(e)) outperforms any individual technique over every single dataset and write variation rate.

[7]The results in Fig. 11 consider the cases in which Swordfish maps only 5% of weights to the SRAM in our RSA-based online retraining approach (see Section 3.4.4). We will revisit this number in Section 5.5.

Fourth, averaged over all the datasets (*Averaged* in Fig. 11-(f)), *Combined* mitigation techniques always produces the highest accuracy on average as well. However, on average, our online *RSA+KD* technique achieves a close accuracy (less than 0.001% difference) for low write variation rates, i.e., write variation less than 10%.)

These results suggest that even with multiple accuracy enhancement techniques, only minor write variations (e.g., less than 10%) can be tolerated. We conclude that a memristor-based CIM-enabeld accelerator for basecalling can be effective even with write variations, but such variations must be kept low (e.g., up to 10%). Fortunately, the projected write variation rate for memristor-based devices [22, 55] suggests the likelihood of achieving this percentage rate. For the rest of this manuscript, we assume a write variation of 10%.

*5.4.2 Effect of Accuracy Enhancement for Combined Non-idealities.* Fig. 12 presents the accuracy of basecalling with different accuracy enhancement techniques in crossbars of 64×64 for the modeled non-idealities. For the non-idealities, we consider the five variations of *Synaptic+Wires*, *Sense+ADC*, *DAC+Driver*, *Combined*, and *Measured* defined in Section 5.2.2. In Fig. 12, we evaluate five accuracy enhancement techniques of *VAT*, *KD*, *R-V-W*, *RSA+KD*, and *All* (as defined in Section 5.4.1) per non-ideality. Fig. 13 presents the same experiments for crossbars of 256×256. As we conclude in Section 5.4, we assume 10% write variation and 5% of the weights are mapped to the SRAM in the online retraining approach (see Section 3.4.4). We present our accuracy results averaged across all the evaluated datasets. We make four main observations from Fig. 12.

(1) Combining of individual accuracy enhancement techniques does not improve the accuracy in an additive manner. For example, each of *VAT*, *R-V-W*, and *RSA+KD* in Fig. 12 improves accuracy due to *Synaptic+Wires* by 6.85%, 10.64%, 10.85%, respectively. However, when we consider all non-idealities together in the *All* configuration, accuracy improves by only 11.84% (❶ in Fig. 12).
(2) The effectiveness of an individual accuracy enhancement technique depends on the underlying error and non-ideality it targets. For example, *VAT* is as effective as *RSA+KD* for non-idealities due to *DAC+Driver* (94.22% vs. 94.32%). However, the gap between the two approaches widens for non-idealities due to *Synaptic+Wires* (87.32% vs. 91.32%). See ❷ in Fig. 12.

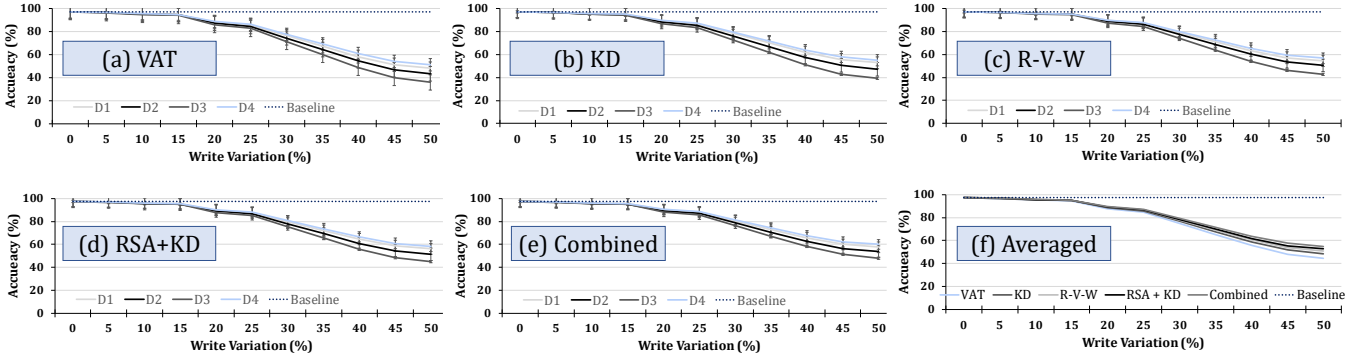


**Figure 11: Accuracy after combining enhancement techniques over different write variations.**

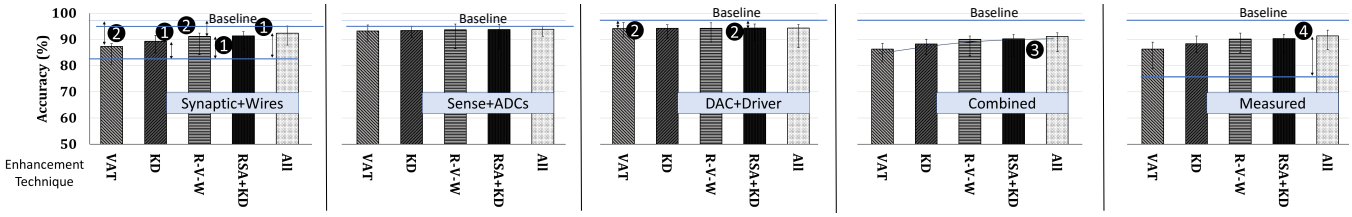


**Figure 12: Accuracy after enhancement mechanisms for evaluated non-idealities on 64×64 crossbars.**

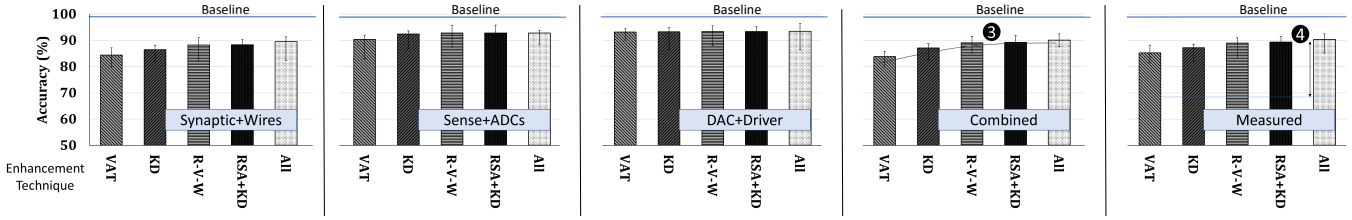


**Figure 13: Accuracy after enhancement mechanisms for evaluated non-idealities on 256×256 crossbars.**

(3) Accuracy enhancement techniques improve accuracy with a similar trend over different crossbar sizes (❸ in Fig. 12 and Fig. 13). Although these results are averaged over our datasets, one can make the same observation on each dataset as well.

(4) Accuracy enhancement techniques are more effective for larger crossbars than for smaller ones (e.g., 256×256 compared to 64×64). This is expected because there is more room for accuracy improvement for these larger crossbars, as their inaccuracies are higher. For example, we observe 22.07% improvement in accuracy for 256×256 crossbars (❹ in Fig. 13) compared to 16.24% for 64×64 (❹ in Fig. 12), after all of the accuracy enhancement techniques are applied (*All*) over all existing non-idealities (i.e., the *Measured* configuration).

We conclude that the basecalling accuracy of SwordfishAccel can match SotA levels by using robust techniques that build on each other employing reasonable crossbar sizes (e.g., 64×64) and successfully accounting for substantial circuit variations, like write variations.

## 5.5 Throughput Analysis of SwordfishAccel

Fig. 14 shows the inference throughput for Bonito on a GPU (Bonito-GPU) card discussed in Section 4.2, Ideal-SwordfishAccel, Realistic-SwordfishAccel-RVW, Realistic-SwordfishAccel-RSA, and Realistic-SwordfishAccel-RSA+KD. We show the results for each of the four datasets and the average results over all datasets. The results are for a crossbar of size 64×64 and a write variation rate of 10%, and assuming 5% of weights are placed in SRAM for Realistic-SwordfishAccel-RSA and Realistic-SwordfishAccel-RSA+KD.

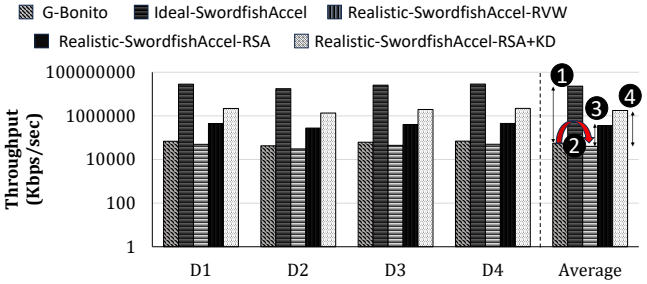


**Figure 14: Throughput comparison of Swordfish variations.**

We make four key observations. First, Ideal-SwordfishAccel improves the basecalling throughput over Bonito-GPU for all datasets, by 413.6× on average (❶ in Fig. 14). We expect such a large improvement in throughput because SwordfishAccel is highly optimized for the main dominant kernel in the underlying DNN of Bonito, namely VMM, and avoids unnecessary data movement while harvesting the maximum parallelism.

Second, all versions of Realistic-SwordfishAccel (i.e., Realistic-SwordfishAccel-RVW, Realistic-SwordfishAccel-RSA, and Realistic-SwordfishAccel-RSA+KD) have lower performance than Ideal--SwordfishAccel, irrespective of the dataset. Performance loss with a realistic Swordfish accelerator is expected because each realistic version adds overheads to mitigate accuracy loss due to realistically-modeled non-idealities, which directly affect the performance of a VMM operation. For example, RSA adds overheads due to (1) the extra checks when reading some weights from the on-chip SRAM memory and (2) additional logic for combining the results from the memristor-based crossbar and on-chip memory readout.

Third, not all versions of Realistic-SwordfishAccel outperform Bonito-GPU. More specifically, if we use R-V-W for mitigating non-idealities (Realistic-SwordfishAccel-RVW in Fig. 14), the overhead due to additional verifications and writes significantly reduces the performance of basecalling throughput compared to Bonito-GPU by 30% on average (❷ in Fig. 14).

Fourth, Realistic-SwordfishAccel-RSA and Realistic-Swordfish-Accel-RSA+KD provide, on average, 5.24× and 25.7× higher throughput compared to Bonito-GPU, respectively (❸ and ❹ in Fig. 14). Note that, for the same accuracy, Realistic-SwordfishAccel-RSA+KD requires fewer weights inside the SRAM than Realistic-Swordfish-Accel-RSA due to the retraining using KD. Hence, Realistic-Sword-fishAccel-RSA+KD is faster.

We conclude that a realistic basecalling accelerator designed using Swordfish by taking into account and mitigating all non-idealities of memristor-based CIM can significantly accelerate basecalling, yet its benefits are much lower than a corresponding accelerator that does not mitigate such non-idealities and thus has much lower accuracy.

## 5.6 Area vs. Accuracy Analysis

Fig. 15 shows the tradeoff between accuracy and area in Realistic-SwordfishAccel-RSA+KD (see Section 5.5) for two different crossbar sizes (64×64 on the left and 256×256 on the right), with four different percentages of weights (i.e., 0%, 1%, 5%, and 10%) assigned to the SRAM memory (see Section 3.4.4). The area numbers show the absolute area for implementing Realistic-SwordfishAccel-RSA+KD considering the overhead of RSA+KD discussed in Section 3.4.4. The red dashed line shows the accuracy of the original Bonito basecaller. We make three main observations.

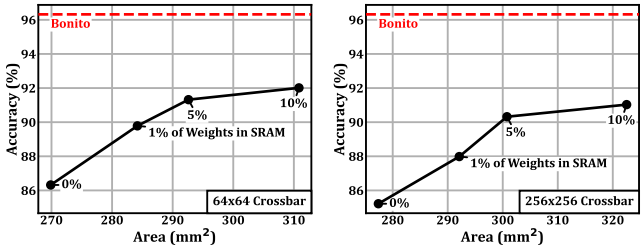


**Figure 15: Accuracy vs. Area evaluation of Realistic-SwordfishAccel-RSA+KD.**

First, the more weights are assigned to SRAM, the higher the accuracy of Realistic-SwordfishAccel-RSA+KD. This is expected because we effectively reduce the non-idealities of the system by using more SRAM cells to remap non-ideal memristors.

Second, the area of extra SRAM cells used in Realistic-SwordfishAccel-RSA+KD increases significantly with the percentage of weights assigned to SRAM. In contrast, the accuracy improvement saturates and does not increase significantly beyond 5% of weights assigned to SRAM.

Third, assigning only 5% of weights to SRAM is sufficient to be within 5% of Bonito-GPU's accuracy for the 64×64 crossbar.

We conclude that accounting for non-idealities in different ways exposes tradeoffs between accuracy and area overhead, which our Swordfish framework enables the designer to rigorously explore.

# 6 Discussion and Future Work

## 6.1 Applicability of Swordfish Looking forward

Swordfish emphasizes the importance of a framework for evaluating multiple metrics when designing a memristor-based CIM accelerator targeting large DNNs that require throughput acceleration while having stringent bound for another metric, e.g., accuracy (in the presence of emerging technologies with many non-idealities).

Swordfish's realistic results, Realistic-SwordfishAccel, for Bonito, a large DNN, challenge the notion that DNN-based applications naturally thrive on memristor-based CIM due to the inherent redundancy present in large neural networks. Although Realistic-SwordfishAccel might not currently offer basecalling accuracy on par with state-of-the-art methods, its large (25.7×) enhancement in performance (Section 5.5) at a much higher accuracy than baseline CIM marks it as an advantageous development. Even in the presence of memristor-based CIM non-ideality, Swordfish still shows promise, and Realistic-SwordfishAccel still maintains a competitive accuracy in basecalling by deploying a unique synergy of mitigation strategies (against non-idealities and variations) on moderately-large crossbar designs (e.g., 64×64 or 256×256). Our results in Section 5 detail this. Given our results, we believe it is productive and important to find more solutions to the memristor-based CIM non-idealities going forward; we believe some solutions will come with memristors becoming more mature, and some will come with more potent accuracy enhancement techniques and HW/SW co-design methods.

## 6.2 Other DNN-based Applications

Our paper discusses Swordfish as a framework for accelerating basecalling using a memristor-based CIM architecture. Our results (Section 5) show the unique nature of the large DNN in Bonito, which, despite its inherent redundancy, does not quite reach SotA accuracy on memristor-based CIM, thus presenting an exciting challenge. This intriguing finding encourages a deeper exploration into CIM designs for large DNNs, reminding us not to rely solely on the scalability assumptions based on small network evaluations, such as simple CNNs for MNIST. Our results also demonstrate a large acceleration opportunity for basecalling using SwordfishAccel if we can mitigate the memristor-induced accuracy loss through HW/SW co-design approaches. We believe other DNN-based applications that use memristor-based CIM accelerators (e.g., [22, 55, 146]) can also benefit from our approach and Swordfish. For example, large DNN models in autonomous driving (e.g., [64, 75, 146]) that require accurate yet high-throughout and low-latency execution can use a Swordfish-like approach to build memristor-based CIM accelerators for their underlying large DNNs. We believe and hope that Swordfish can aid such applications in terms of both accuracy and performance.

## 6.3 Better Accuracy Enhancement Techniques

Our results show that accuracy enhancement can pave the way toward SwordfishAccel becoming a reliable solution. Our online retraining mechanism shows the highest potential to improve the accuracy loss. We believe there needs to be more research on better mitigation techniques for existing and future non-idealities in memristor-based designs. Specifically, we suggest hardware/software co-design solutions such as our RSA+KD technique in Section 3.4.4. Hardware-based solutions to mitigate non-idealities [25] that are orthogonal to our RSA+KD approach is also an example of possible avenues of future work.

# 7 Related Work

To our knowledge, Swordfish is the first framework that enables evaluating the acceleration of large Deep Neural Networks (DNNs) on memristor-based Computation-In-Memory (CIM) designs considering hardware non-idealities. We have already compared Swordfish extensively to the currently-used version of the Bonito basecaller in Section 5 in terms of accuracy, throughput, and area overhead. This section briefly discusses related prior works on basecallers and CIM accelerators.

## 7.1 Genomic Basecallers

Several recent works propose approaches and techniques to either improve the accuracy of basecalling or accelerate it with minimum accuracy loss. These works take three main approaches: (1) new DNN architectures (e.g., [92, 95–97, 120, 134, 140]), (2) new

hardware platforms and designs such as GPUs and FPGAs to execute previously-proposed basecallers with minimum modifications (e.g., [81, 120]), and (3) software techniques such as quantization to reduce the computation and storage overhead (e.g., [32, 35, 52, 74, 120, 124, 140]).

In contrast to these approaches, Swordfish is a framework for the *evaluation* of DNN-based (basecalling) accelerators. As such, Swordfish is orthogonal to prior works in basecalling, enabling proper evaluation of relevant works in the context of memristor-based in-memory acceleration.

### 7.2 Computation-In-Memory Accelerators

Many previous works investigate how to provide new functionality using compute-capable memories based on conventional (e.g., [1, 2, 36, 40, 43, 72, 99, 108, 110]) and emerging memory technologies (e.g., [9, 27, 56, 59, 68, 73, 101, 111, 116, 117, 119, 121, 139, 144]) to help solve the data movement overheads in today's systems. These works propose new functionality in at least three major categories: (1) support for logical operations (e.g., [26, 73, 86, 110, 119, 121, 139, 144]), (2) support for complex operations, functions, and applications (e.g., [1, 36, 72, 89, 111, 112, 116, 117, 143]), and (3) programming and system support for the integration and adoption of such accelerators (e.g., [2, 3, 9, 19, 27, 55, 86, 111, 144, 145]).

Several prior works(e.g., [22, 51, 55, 128]) investigate the new requirements, tradeoffs, and challenges that arise from using the CIM paradigm (e.g., dealing with non-idealities in the analog operations). To our knowledge, no work has proposed a complete solution or framework for these challenges; thus, this area requires further investigation.

Swordfish aligns with these works as it provides (1) new functionality for compute-capable memristors at the application level for accelerating genomic basecalling and (2) a framework for evaluating the practical challenges posed by the non-idealities in the memristor computation through mitigation techniques.

## 8 Conclusion

This paper introduces Swordfish, a modular and extensible framework for accelerating the evaluation of genomic basecalling via a memristor-based Computation-In-Memory architecture. Swordfish includes a strong evaluation methodology, mitigation strategies for hardware non-idealities, and characterization results to guide the modeling of memristors. Using Swordfish, we demonstrate the significant challenges of using non-ideal memristor-based computations for genomic basecalling and how to solve them by combining multiple mitigation techniques at the circuit and system levels. We demonstrate the usefulness of our findings by developing SwordfishAccel, a concrete memristor-based CIM design for our target basecaller Bonito that uses accuracy enhancement techniques guided by Swordfish. We conclude that the Swordfish framework effectively facilitates the development and adoption of memristor-based CIM designs for basecalling, which we hope will be leveraged by future work. We also believe that our framework is applicable to other DNN-based applications and hope future work takes advantage of this.

## Acknowledgments

We thank the anonymous reviewers of MICRO 2023 for their valuable feedback. We thank the members of the QCE department at TU Delft and the SAFARI Research Group at ETH Zurich for valuable feedback and the stimulating intellectual environment they provide. We acknowledge the generous gifts provided by our industrial partners, including Google, Huawei, Intel, Microsoft, and VMware. This research was partially supported by the EU Horizon project BioPIM (grant agreement 101047160), the AI Chip Center for Emerging Smart Systems Limited (ACCESS), the Swiss National Science Foundation (SNSF), Semiconductor Research Corporation (SRC), and the ETH Future Computing Laboratory (EFCL).

## References

[1] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna Das. Compute Caches. In *HPCA*. 2017.
[2] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing. In *ISCA*. 2015.
[3] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. PIM-enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture. In *ISCA*. 2015.
[4] Fabien Alibart, Ligang Gao, Brian D Hoskins, and Dmitri B Strukov. High Precision Tuning of State for Memristive Devices by Adaptable Variation-Tolerant Algorithm. *Nanotechnology*. 2012.
[5] Can Alkan, Jeffrey M Kidd, Tomas Marques-Bonet, Gozde Aksay, Francesca Antonacci, Fereydoun Hormozdiari, et al. Personalized Copy Number and Segmental Duplication Maps Using Next-Generation Sequencing. *Nature Genetics*. 2009.
[6] Mohammed Alser, Joel Lindegger, Can Firtina, Nour Almadhoun, Haiyu Mao, Gagandeep Singh, et al. From Molecules to Genomic Variations: Accelerating Genome Analysis via Intelligent Algorithms and Architectures. *Computational and Structural Biotechnology Journal*. 2022.
[7] Maria Jesus Alvarez-Cubero, Maria Saiz, Belén Martínez-García, Sara M Sayalero, Carmen Entrala, Jose Antonio Lorente, et al. Next Generation Sequencing: An Application in Forensic Sciences? *Annals of Human Biology*. 2017.
[8] AMD. AMD EPYC 7742 CPU. https://www.amd.com/en/products/cpu/amd-epyc-7742.
[9] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R Stanley Williams, et al. PUMA: A Programmable Ultra-Efficient Memristor-Based Accelerator for Machine Learning Inference. In *ASPLOS*. 2019.
[10] Ankit, Aayush. PUMA Compiler. https://github.com/Aayush-Ankit/puma-compiler.
[11] Ankit, Aayush and Kim, Dong-Eun and Chakraborty, Indranil and Ali, Mustafa and Negi, Shubham. PUMA Functional Simulator. https://github.com/Aayush-Ankit/puma-functional-model.
[12] Simon Ardui, Adam Ameur, Joris R Vermeesch, and Matthew S Hestand. Single Molecule Real-Time (SMRT) Sequencing Comes of Age: Applications and Utilities for Medical Diagnostics. *Nucleic Acids Research*. 2018.
[13] Zahra Aryan, Attila Szanto, Angeliki Pantazi, Tejaswini Reddi, Carolyn Rheinstein, Winslow Powers, et al. Moving Genomics to Routine Care: An Initial Pilot in Acute Cardiovascular Disease. *Circulation: Genomic and Precision Medicine*. 2020.
[14] Euan A Ashley. Towards Precision Medicine. *Nature Reviews Genetics*. 2016.
[15] Joshua S Bloom, Laila Sathe, Chetan Munugala, Eric M Jones, Molly Gasperini, Nathan B Lubock, et al. Massively Scaled-Up Testing for SARS-CoV-2 RNA via Next-Generation Sequencing of Pooled and Barcoded Nasal and Saliva Samples. *Nature Biomedical Engineering*. 2021.
[16] Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F. Oliveira, Xiaoyu Ma, et al. Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks. In *PACT*. 2021.
[17] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, et al. Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks. In *ASPLOS*. 2018.
[18] Daniel Branton, David W Deamer, Andre Marziali, Hagan Bayley, Steven A Benner, Thomas Butler, et al. The Potential and Challenges of Nanopore Sequencing. *Nature Biotechnology*. 2008.
[19] Geoffrey W Burr, Robert M Shelby, Abu Sebastian, Sangbum Kim, Seyoung Kim, Severin Sidler, et al. Neuromorphic Computing Using Non-Volatile Memory. *Advances in Physics: X*. 2017.
[20] Damla Senol Cali, Gurpreet S Kalsi, Zülal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S Kim, et al. GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis. In *MICRO*. 2020.
[21] Kevin K. Chang, Prashant J. Nair, Donghyuk Lee, Saugata Ghose, Moinuddin K. Qureshi, and Onur Mutlu. Low-Cost Inter-Linked Subarrays (LISA): Enabling

Fast Inter-Subarray Data Movement in DRAM. In *HPCA*. 2016.
[22] Gouranga Charan, Abinash Mohanty, Xiaocong Du, Gokul Krishnan, Rajiv V Joshi, and Yu Cao. Accurate Inference With Inaccurate RRAM Devices: A Joint Algorithm-Design Solution. *JXCDC*. 2020.
[23] Ching-Yi Chen, Hsiu-Chuan Shih, Cheng-Wen Wu, Chih-He Lin, Pi-Feng Chiu, Shyh-Shyuan Sheu, et al. RRAM Defect Modeling and Failure Analysis Based on March Test and a Novel Squeeze-Search Scheme. *IEEE Transactions on Computers*. 2014.
[24] Lerong Chen, Jiawen Li, Yiran Chen, Qiuping Deng, Jiyuan Shen, Xiaoyao Liang, et al. Accelerator-Friendly Neural-Network Training: Learning Variations and Defects in RRAM Crossbar. In *DATE*. 2017.
[25] Lerong Chen, Jiawen Li, Yiran Chen, Qiuping Deng, Jiyuan Shen, Xiaoyao Liang, et al. Accelerator-Friendly Neural-Network Training: Learning Variations and Defects in RRAM Crossbar. In *DATE*. 2017.
[26] Long Cheng, Yi Li, Kang-Sheng Yin, Si-Yu Hu, Yu-Ting Su, Miao-Miao Jin, et al. Functional Demonstration of a Memristive Arithmetic Logic Unit (MemALU) for In-Memory Computing. *Advanced Functional Materials*. 2019.
[27] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, et al. PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. *ISCA*. 2016.
[28] Lynda Chin, Jannik N Andersen, and P Andrew Futreal. Cancer Genomics: From Discovery Science to Personalized Medicine. *Nature Medicine*. 2011.
[29] Michelle M Clark, Amber Hildreth, Sergey Batalov, Yan Ding, Shimul Chowdhury, Kelly Watkins, et al. Diagnosis of Genetic Diseases in Seriously Ill Children by Rapid Whole-Genome Sequencing and Automated Phenotyping and Interpretation. *Science Translational Medicine*. 2019.
[30] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training Deep Neural Networks With Binary Weights During Propagations. *NeurIPS*. 2015.
[31] Raquel Dias and Ali Torkamani. Artificial Intelligence in Clinical and Genomic Diagnostics. *Genome Medicine*. 2019.
[32] Ruizhou Ding, Zeye Liu, Rongye Shi, Diana Marculescu, and RD Blanton. LightNN: Filling the Gap Between Conventional Deep Neural Networks and Binarized Networks. In *GLSVLSI*. 2017.
[33] Tim Dunn, Harisankar Sadasivan, Jack Wadden, Kush Goliya, Kuan-Yu Chen, David Blaauw, et al. SquiggleFilter: An Accelerator for Portable Virus Detection. In *MICRO*. 2021.
[34] Hans Ellegren. Genome Sequencing and Population Genomics in Non-Model Organisms. *Trends in Ecology & Evolution*. 2014.
[35] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned Step Size Quantization. *arXiv*. 2019.
[36] João Dinis Ferreira, Gabriel Falcao, Juan Gómez-Luna, Mohammed Alser, Lois Orosa, Mohammad Sadrosadati, et al. pLUTo: In-DRAM Lookup Tables to Enable Massively Parallel General-Purpose Computation. *MICRO*. 2022.
[37] Markus Fritscher, Johannes Knödtel, Maen Mallah, Stefan Pechmann, Emilio Perez-Bosch Quesada, Tommaso Rizzi, et al. Mitigating the Effects of RRAM Process Variation on the Accuracy of Artificial Neural Networks. In *International Conference on Embedded Computer Systems*. 2022.
[38] Markus Fritscher, Johannes Knödtel, Daniel Reiser, Maen Mallah, Stefan Pechmann, Dietmar Fey, et al. Simulating Large Neural Networks Embedding MLC RRAM as Weight Storage Considering Device Variations. In *LASCAS*. 2021.
[39] Daichi Fujiki, Arun Subramaniyan, Tianjun Zhang, Yu Zeng, Reetuparna Das, David Blaauw, et al. GenAx: A Genome Sequencing Accelerator. In *ISCA*. 2018.
[40] Congming Gao, Xin Xin, Youyou Lu, Youtao Zhang, Jun Yang, and Jiwu Shu. ParaBit: Processing Parallel Bitwise Operations in NAND Flash Memory based SSDs. In *MICRO*. 2021.
[41] Geoffrey S Ginsburg and Kathryn A Phillips. Precision Medicine: From Science to Value. *Health Affairs*. 2018.
[42] Geoffrey S Ginsburg and Huntington F Willard. Genomic and Personalized Medicine: Foundations and Applications. *Translational Research*. 2009.
[43] Juan Gómez-Luna, Yuxin Guo, Sylvan Brocard, Julien Legriel, Remy Cimadomo, Geraldo F Oliveira, et al. An Experimental Evaluation of Machine Learning Training on a Real Processing-in-Memory System. *arXiv preprint arXiv:2207.07886*. 2022.
[44] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep Learning With Limited Numerical Precision. In *ICML*. 2015.
[45] Said Hamdioui, Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Koen Bertels, Henk Corporaal, et al. Memristor Based Computation-In-Memory Architecture for Data-Intensive Applications. In *DATE*. 2015.
[46] Song Han, Huizi Mao, and William J Dally. Deep Compression: Compressing Deep Neural Networks With Pruning, Trained Quantization and Huffman Coding. *ICLR*. 2015.
[47] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv*. 2015.
[48] Miao Hu, Catherine E Graves, Can Li, Yunning Li, Ning Ge, Eric Montgomery, et al. Memristor-Based Analog Computation and Neural Network Classification With a Dot Product Engine. *Advanced Materials*. 2018.
[49] Miao Hu, Hai Li, Yiran Chen, Qing Wu, and Garrett S Rose. BSB Training Scheme Implementation on Memristor-Based Circuit. In *CISDA*. 2013.
[50] Neng Huang, Fan Nie, Peng Ni, Feng Luo, and Jianxin Wang. SACall: A Neural Network Basecaller for Oxford Nanopore Sequencing Data Based on Self-Attention Mechanism. *TCBB*. 2020.
[51] Ahmet Inci, Siri Garudanagiri Virupaksha, Aman Jain, Ting-Wu Chin, Venkata Vivek Thallam, Ruizhou Ding, et al. QUIDAM: A Framework for Quantization DNN Accelerator and Model Co-Exploration. *TECS*. 2022.
[52] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, et al. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *CVPR*. 2018.
[53] Miten Jain, Sergey Koren, Karen H Miga, Josh Quick, Arthur C Rand, Thomas A Sasani, et al. Nanopore Sequencing and Assembly of a Human Genome With Ultra-Long Reads. *Nature Biotechnology*. 2018.
[54] Shubham Jain and Anand Raghunathan. CxDNN: Hardware-Software Compensation Methods for Deep Neural Networks on Resistive Crossbar Systems. *TECS*. 2019.
[55] Shubham Jain, Abhronil Sengupta, Kaushik Roy, and Anand Raghunathan. RxNN: A Framework for Evaluating Deep Neural Networks on Resistive Crossbars. *TCAD*. 2020.
[56] YeonJoo Jeong, Mohammed A Zidan, and Wei D Lu. Parasitic Effect Analysis in Memristor-Array-Based Neuromorphic Systems. *IEEE Transactions on Nanotechnology*. 2017.
[57] Weiwen Jiang, Qiuwen Lou, Zheyu Yan, Lei Yang, Jingtong Hu, Xiaobo Sharon Hu, et al. Device-Circuit-Architecture Co-Exploration for Computing-in-Memory Neural Accelerators. *TC*. 2020.
[58] VG Karpov, YA Kryukov, SD Savransky, and IV Karpov. Nucleation Switching in Phase Change Memory. *Applied Physics Letters*. 2007.
[59] Geethan Karunaratne, Manuel Le Gallo, Giovanni Cherubini, Luca Benini, Abbas Rahimi, and Abu Sebastian. In-Memory Hyperdimensional Computing. *Nature Electronics*. 2020.
[60] Mehdi Kchouk, Jean-Francois Gibrat, and Mourad Elloumi. Generations of Sequencing Technologies: From First to Next Generation. *Biology and Medicine*. 2017.
[61] Wonjoo Kim, Anupam Chattopadhyay, Anne Siemon, Eike Linn, Rainer Waser, and Vikas Rana. Multistate Memristive Tantalum Oxide Devices for Ternary Arithmetic. *Scientific Reports*. 2016.
[62] Stephen F Kingsmore, Laurie D Smith, Chris M Kunard, Matthew Bainbridge, Sergey Batalov, Wendy Benson, et al. A Genome Sequencing System for Universal Newborn Screening, Diagnosis, and Precision Medicine for Severe Genetic Diseases. *The American Journal of Human Genetics*. 2022.
[63] Michael Klachko, Mohammad Reza Mahmoodi, and Dmitri Strukov. Improving Noise Tolerance of Mixed-Signal Neural Networks. In *IJCNN*. 2019.
[64] Jelena Kocić, Nenad Jovičić, and Vujo Drndarević. An End-to-End Deep Neural Network for Autonomous Driving Designed for Embedded Automotive Platforms. *Sensors*. 2019.
[65] Hiroki Konishi, Rui Yamaguchi, Kiyoshi Yamaguchi, Yoichi Furukawa, and Seiya Imoto. Halcyon: An Accurate Basecaller Exploiting an Encoder–Decoder Model With Monotonic Attention. *Bioinformatics*. 2021.
[66] Skanda Koppula, Lois Orosa, A Giray Yağlıkçı, Roknoddin Azizi, Taha Shahroodi, Konstantinos Kanellopoulos, et al. EDEN: Enabling Energy-Efficient, High-Performance Deep Neural Network Inference Using Approximate DRAM. In *MICRO*. 2019.
[67] Vien Thi Minh Le and Binh An Diep. Selected Insights From Application of Whole Genome Sequencing for Outbreak Investigations. *Current Opinion in Critical Care*. 2013.
[68] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *ISCA*. 2009.
[69] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Phase Change Memory Architecture and the Quest for Scalability. *CACM*. 2010.
[70] Jung-Hoon Lee, Dong-Hyeok Lim, Hongsik Jeong, Huimin Ma, and Luping Shi. Exploring Cycle-to-Cycle and Device-to-Device Variation Tolerance in MLC Storage-Based Neural Network Training. *IEEE Transactions on Electron Devices*. 2019.
[71] Heng Li. Minimap and Miniasm: Fast Mapping and De Novo Assembly for Noisy Long Sequences. *Bioinformatics*. 2016.
[72] Shuangchen Li, Dimin Niu, Krishna T Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. DRISA: A DRAM-Based Reconfigurable In-Situ Accelerator. In *MICRO*. 2017.
[73] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-Volatile Memories. In *DAC*. 2016.
[74] Yuhang Li, Xin Dong, and Wei Wang. Additive Powers-of-Two Quantization: An Efficient Non-Uniform Discretization for Neural Networks. *arXiv*. 2019.
[75] Zhong Li, Minxue Pan, Tian Zhang, and Xuandong Li. Testing DNN-based Autonomous Driving Systems under Critical Environmental Conditions. In *ICML*. 2021.

[76] Meng-Yao Lin, Hsiang-Yun Cheng, Wei-Ting Lin, Tzu-Hsien Yang, I-Ching Tseng, Chia-Lin Yang, et al. DL-RSIM: A Simulation Framework to Enable Reliable ReRAM-Based Accelerators for Deep Learning. In *ICCAD*. 2018.

[77] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Tingwen Huang, Qing Wu, et al. Reduction and IR-drop Compensations Techniques for Reliable Neuromorphic Computing Systems. In *ICCAD*. 2014.

[78] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Qing Wu, and Tingwen Huang. Vortex: Variation-Aware Training for Memristor X-Bar. In *DAC*. 2015.

[79] Jiawen Liu, Hengyu Zhao, Matheus A Ogleari, Dong Li, and Jishen Zhao. Processing-in-Memory for Energy-Efficient Neural Network Training: A Heterogeneous Approach. In *MICRO*. 2018.

[80] Yun Long, Xueyuan She, and Saibal Mukhopadhyay. Design of Reliable DNN Accelerator With Un-Reliable ReRAM. In *DATE*. 2019.

[81] Qian Lou, Sarath Chandra Janga, and Lei Jiang. Helix: Algorithm/Architecture Co-Design for Accelerating Nanopore Genome Base-Calling. In *PACT*. 2020.

[82] MNEMOSENE partners. The MNEMOSENE Project. http://www.mnemosene.eu/.

[83] Mohamad G. Moinuddin, Aijaz H. Lone, Shivangi Shringi, Srikant Srinivasan, and Satinder K. Sharma. Low-Current-Density Magnetic Tunnel Junctions for STT-RAM Application Using MgO $_x$ N $_{1-x}$ ($x = 0.57$ ) Tunnel Barrier. *IEEE Transactions on Electron Devices*. 2020.

[84] Onur Mutlu and Can Firtina. Accelerating Genome Analysis via Algorithm-Architecture Co-Design. In *DAC*. 2023.

[85] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. A Modern Primer on Processing in Memory. In *Emerging Computing: From Devices to Systems*. 2023.

[86] Leibin Ni, Zichuan Liu, Hao Yu, and Rajiv V Joshi. An Energy-Efficient Digital ReRAM-Crossbar-Based CNN With Bitwise Parallelism. *JXCDC*. 2017.

[87] Vlad Nikolayevskyy, Katharina Kranzer, Stefan Niemann, and Francis Drobniewski. Whole Genome Sequencing of Mycobacterium Tuberculosis for Detection of Recent Transmission and Tracing Outbreaks: A Systematic Review. *Tuberculosis*. 2016.

[88] NVIDIA. NVIDIA V100. https://www.nvidia.com/en-us/data-center/v100/.

[89] Geraldo F Oliveira, Juan Gómez-Luna, Mohammad Sadrosadati, Yuxin Guo, and Onur Mutlu. TransPimLib: Efficient Transcendental Functions for Processing-in-Memory Systems. In *ISPASS*. 2023.

[90] Oxford Nanopore Technologies Ltd. Developers. GridION. https://nanoporetech.com/products/gridion. 2017.

[91] Oxford Nanopore Technologies Ltd. Developers. Metrichor. https://metrichor.com. 2017.

[92] Oxford Nanopore Technologies Ltd. Developers. Flappie. https://github.com/nanoporetech/flappie. 2018.

[93] Oxford Nanopore Technologies Ltd. Developers. PromethION. https://nanoporetech.com/products/promethion-2. 2018.

[94] Oxford Nanopore Technologies Ltd. Developers. MinION. https://nanoporetech.com/products/minion. 2019.

[95] Oxford Nanopore Technologies Ltd. Developers. Scrappie. https://github.com/nanoporetech/scrappie. 2019.

[96] Oxford Nanopore Technologies Ltd. Developers. Bonito. https://github.com/nanoporetech/bonito. 2020.

[97] Oxford Nanopore Technologies Ltd. Developers. Dorado. https://github.com/nanoporetech/dorado. 2022.

[98] Marc Pages-Gallego and Jeroen de Ridder. Comprehensive and Standardized Benchmarking of Deep Learning Architectures for Basecalling Nanopore Sequencing Data. *bioRxiv*. 2022.

[99] Jisung Park, Roknoddin Azizi, Geraldo F Oliveira, Mohammad Sadrosadati, Rakesh Nadig, David Novo, et al. Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory. In *MICRO*. 2022.

[100] Giacomo Pedretti, Elia Ambrosi, and Daniele Ielmini. Conductance Variations and Their Impact on the Precision of In-Memory Computing With Resistive Switching Memory (RRAM). In *IRPS*. 2021.

[101] Mirko Prezioso, Farnood Merrikh-Bayat, Brian D Hoskins, Gina C Adam, Konstantin K Likharev, and Dmitri B Strukov. Training and Operation of an Integrated Neuromorphic Network Based on Metal-Oxide Memristors. *Nature*. 2015.

[102] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary Neural Networks: A Survey. *Pattern Recognition*. 2020.

[103] Joshua Quick, Nicholas J Loman, Sophie Duraffour, Jared T Simpson, Ettore Severi, Lauren Cowley, et al. Real-Time, Portable Genome Sequencing for Ebola Surveillance. *Nature*. 2016.

[104] S. Ramachandran, J.W. Nicholson, S. Ghalmi, and M.F. Yan. Measurement of Multipath Interference in the Coherent Crosstalk Regime. *IEEE Photonics Technology Letters*. 2003.

[105] Franka J Rang, Wigard P Kloosterman, and Jeroen de Ridder. From Squiggle to Basepair: Computational Approaches for Improving Nanopore Sequencing Read Accuracy. *Genome Biology*. 2018.

[106] Satyabrata Sarangi and Bevan Baas. DeepScaleTool: A Tool for the Accurate Estimation of Technology Scaling in the Deep-Submicron Era. In *ISCAS*. 2021.

[107] Damla Senol Cali, Jeremie S Kim, Saugata Ghose, Can Alkan, and Onur Mutlu. Nanopore Sequencing Technology and Tools for Genome Assembly: Computational Analysis of the Current State, Bottlenecks and Future Directions. *Briefings in Bioinformatics*. 2019.

[108] Vivek Seshadri, Kevin Hsieh, Amirali Boroum, Donghyuk Lee, Michael A Kozuch, Onur Mutlu, et al. Fast Bulk Bitwise AND and OR in DRAM. *CAL*. 2015.

[109] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, et al. RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization. In *MICRO*. 2013.

[110] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, et al. Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology. In *MICRO*. 2017.

[111] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, et al. ISAAC: A Convolutional Neural Network Accelerator With In-Situ Analog Arithmetic in Crossbars. *ISCA*. 2016.

[112] Taha Shahroodi, Raphael Cardoso, Mahdi Zahedi, Stephan Wong, Alberto Bosio, Ian O'Connor, et al. Lightspeed Binary Neural Networks Using Optical Phase-Change Materials. In *DATE*. 2023.

[113] Taha Shahroodi, Michael Miao, Mahdi Zahedi, Stephan Wong, and Said Hamdioui. RattlesnakeJake: A Fast and Accurate Pre-Alignment Filter Suitable for Computation-In-Memory. In *SAMOS*. 2023.

[114] Taha Shahroodi, Michael Miao, Mahdi Zahedi, Stephan Wong, and Said Hamdioui. SieveMem: A Computation-In-Memory Architecture for Fast and Accurate Pre-Alignment. In *ASAP*. 2023.

[115] Taha Shahroodi, Stephan Wong, and Said Hamdioui. A Case for Genome Analysis Where Genomes Reside. In *SAMOS*. 2023.

[116] Taha Shahroodi, Mahdi Zahedi, Can Firtina, Mohammed Alser, Stephan Wong, Onur Mutlu, et al. Demeter: A Fast and Energy-Efficient Food Profiler Using Hyperdimensional Computing in Memory. *IEEE Access*. 2022.

[117] Taha Shahroodi, Mahdi Zahedi, Abhairaj Singh, Stephan Wong, and Said Hamdioui. KrakenOnMem: A Memristor-Augmented HW/SW Framework for Taxonomic Profiling. In *ICS*. 2022.

[118] Lingyun Shi, Guohao Zheng, Bobo Tian, Brahim Dkhil, and Chungang Duan. Research Progress on Solutions to the Sneak Path Issue in Memristor Crossbar Arrays. *Nanoscale Advances*. 2020.

[119] Abhairaj Singh, Mahdi Zahedi, Taha Shahroodi, Mohit Gupta, Anteneh Gebregiorgis, Manu Komalan, et al. CIM-Based Robust Logic Accelerator Using 28 nm STT-MRAM Characterization Chip Tape-Out. In *AICAS*. 2022.

[120] Gagandeep Singh, Mohammed Alser, Alireza Khodamoradi, Kristof Denolf, Can Firtina, Meryem Banu Cavlak, et al. A Framework for Designing Efficient Deep Learning-Based Genomic Basecallers. *arXiv*. 2022.

[121] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The Missing Memristor Found. *Nature*. 2008.

[122] Synopsys, Inc. Synopsys Design Compiler. https://www.synopsys.com/support/training/rtl-synthesis/design-compiler-rtl-synthesis.html.

[123] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *AAAI*. 2017.

[124] Thierry Tambe, En-Yu Yang, Zishen Wan, Yuntian Deng, Vijay Janapa Reddi, Alexander Rush, et al. Algorithm-Hardware Co-Design of Adaptive Floating-Point Encodings for Resilient Deep Learning Inference. In *DAC*. 2020.

[125] Hokchhay Tann, Soheil Hashemi, R Iris Bahar, and Sherief Reda. Hardware-Software Codesign of Accurate, Multiplier-Free Deep Neural Networks. In *DAC*. 2017.

[126] Kodai Ueyoshi, Kota Ando, Kazutoshi Hirose, Shinya Takamaeda-Yamazaki, Junichiro Kadomoto, Tomoki Miyata, et al. QUEST: A 7.49 TOPS Multi-Purpose Log-Quantized DNN Inference Engine Stacked on 96MB 3D SRAM Using Inductive-Coupling Technology in 40nm CMOS. In *ISSCC*. 2018.

[127] Erwin L Van Dijk, Yan Jaszczyszyn, Delphine Naquin, and Claude Thermes. The Third Revolution in Sequencing Technology. *Trends in Genetics*. 2018.

[128] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. AxNN: Energy-Efficient Neuromorphic Systems Using Approximate Computing. In *ISLPED*. 2014.

[129] A. Vittal, L.H. Chen, M. Marek-Sadowska, Kai-Ping Wang, and S. Yang. Crosstalk in VLSI Interconnections. *TCAD*. 1999.

[130] A. Vittal and M. Marek-Sadowska. Crosstalk Reduction for VLSI. *TCAD*. 1997.

[131] Yunhao Wang, Yue Zhao, Audrey Bollas, Yuru Wang, and Kin Fai Au. Nanopore Sequencing Technology, Bioinformatics and Applications. *Nature Biotechnology*. 2021.

[132] Rainer Waser, Regina Dittmann, Georgi Staikov, and Kristof Szot. Redox-Based Resistive Switching Memories–Nanoionic Mechanisms, Prospects, and Challenges. *Advanced Materials*. 2009.

[133] Jason L Weirather, Mariateresa de Cesare, Yunhao Wang, Paolo Piazza, Vittorio Sebastiano, Xiu-Jie Wang, et al. Comprehensive Comparison of Pacific Biosciences and Oxford Nanopore Technologies and Their Applications to Transcriptome Analysis. *F1000Research*. 2017.

[134] Ryan R Wick, Louise M Judd, and Kathryn E Holt. Performance of Neural Network Basecalling Tools for Oxford Nanopore Sequencing. *Genome Biology*. 2019.

[135] Wick, Ryan. Raw FAST5s. https://bridges.monash.edu/articles/dataset/Raw_fast5s/7676174.

[136] Wick, Ryan. Reference Genomes. https://bridges.monash.edu/articles/dataset/Reference_genomes/7676135.

[137] John C Wooley, Adam Godzik, and Iddo Friedberg. A Primer on Metagenomics. *PLoS Computational Biology*. 2010.

[138] Qiangfei Xia and J Joshua Yang. Memristive Crossbar Arrays for Brain-Inspired Computing. *Nature Materials*. 2019.

[139] Lei Xie, Hoang Anh Du Nguyen, Jintao Yu, Ali Kaichouhi, Mottaqiallah Taouil, Mohammad AlFailakawi, et al. Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing. In *ISVLSI*. 2017.

[140] Zhimeng Xu, Yuting Mai, Denghui Liu, Wenjun He, Xinyuan Lin, Chi Xu, et al. Fast-Bonito: A Faster Deep Learning Based Basecaller for Nanopore Sequencing. *Artificial Intelligence in the Life Sciences*. 2021.

[141] Charlene Yang. Hierarchical Roofline Analysis: How to Collect Data Using Performance Tools on Intel CPUs and NVIDIA GPUs. *arXiv*. 2020.

[142] Ramesh Yelagandula, Aleksandr Bykov, Alexander Vogt, Robert Heinen, Ezgi Özkan, Marcus Martin Strobl, et al. Multiplexed Detection of SARS-CoV-2 and Other Respiratory Infections in High Throughput by SARSeq. *Nature Communications*. 2021.

[143] Mahdi Zahedi, Geert Custers, Taha Shahroodi, Georgi Gaydadjiev, Stephan Wong, and Said Hamdioui. SparseMEM: Energy-Efficient Design for In-Memory Sparse-Based Graph Processing. In *DATE*. 2023.

[144] Mahdi Zahedi, Taha Shahroodi, Geert Custers, Abhairaj Singh, Stephan Wong, and Said Hamdioui. System Design for Computation-In-Memory: From Primitive to Complex Functions. In *VLSI-SoC*. 2022.

[145] Mahdi Zahedi, Taha Shahroodi, Stephan Wong, and Said Hamdioui. Efficient Signed Arithmetic Multiplication on Memristor-Based Crossbar. *IEEE Access*. 2023.

[146] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. DeepRoad: GAN-based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 2018.

[147] Wenqiang Zhang, Xiaochen Peng, Huaqiang Wu, Bin Gao, Hu He, Youhui Zhang, et al. Design Guidelines of RRAM Based Neural-Processing-Unit: A Joint Device-Circuit-Algorithm Analysis. In *DAC*. 2019.

[148] Yaojun Zhang, Xiaobin Wang, Hai Li, and Yiran Chen. STT-RAM Cell Optimization Considering MTJ and CMOS Variations. *IEEE Transactions on Magnetics*. 2011.

[149] Yao-zhong Zhang, Arda Akdemir, Georg Tremmel, Seiya Imoto, Satoru Miyano, Tetsuo Shibuya, et al. Nanopore Basecalling From a Perspective of Instance Segmentation. *BMC Bioinformatics*. 2020.