Swordfish:

A Framework for Evaluating Deep Neural Network-based Basecalling using Computation-in-Memory with Non-Ideal Memristors

Taha Michael Shahroodi,
Gagandeep Singh, Mahdi Zahedi, Haiyu Mao,
Joel Lindegger, Can Firtina, Stephan Wong,
Onur Mutlu, Said Hamdioui

Quantum & Computer Engineering
SAFARI Research Group
safari.ethz.ch
TU Delft
Delft University of Technology
AMD
ETH zürich

Authors

Tara Shahroodi

PhD DUT

Quant Researcher

Joel Lindegger

PhD Student @ ETH

Stephan Wong

PhD DUT

DUT Prof.

Gagandeep Singh

PhD ETH

UIUC Prof

Can Firtina

Senior Researcher @ PhD @ ETH

Onur Mutlu

PhD UT Austin

ETH/CMU Prof.

Mahdi Zahedi

PhD DUT

Hardware Engineer @ CERN

Haiyu Mao

PhD Tsinghua

KCL Lecturer, Postdoc @ ETH

Said Hamdioui

PhD DUT

DUT Prof.

Outline

Background & Motivation

Swordfish: Design & Implementation

Evaluation & Key Results

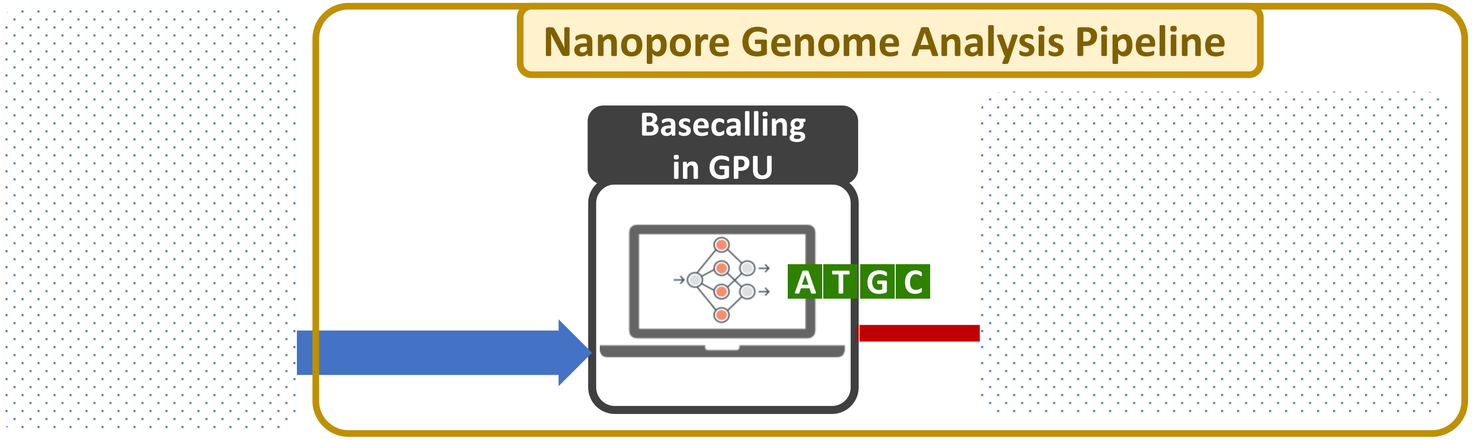
Takeaways & Summary

Nanopore Genome Sequencing and Analysis Pipeline

Genome Sequencing: Determining DNA sequence order for

1. Personalized medicine,
2. Outbreak tracing,
3. Understanding evolution

Nanopore Sequencing: A widely used sequencing technology



Basecalling consumes **up to 84.2%** of the execution time **[Bowden+ 2019]**

Nanopore Genome Sequencing and Analysis Pipeline

Genome Sequencing: Determining DNA sequence order for

1. Personalized medicine,
2. Outbreak tracing,
3. Understanding evolution

Nanopore Sequencing: A widely used sequencing technology

Basecalling is

1. **Accuracy-critical**
2. **Performance Bottleneck**

Basecallers are just large DNNs

DNN Hardware Acceleration

DNN execution is dominated by:

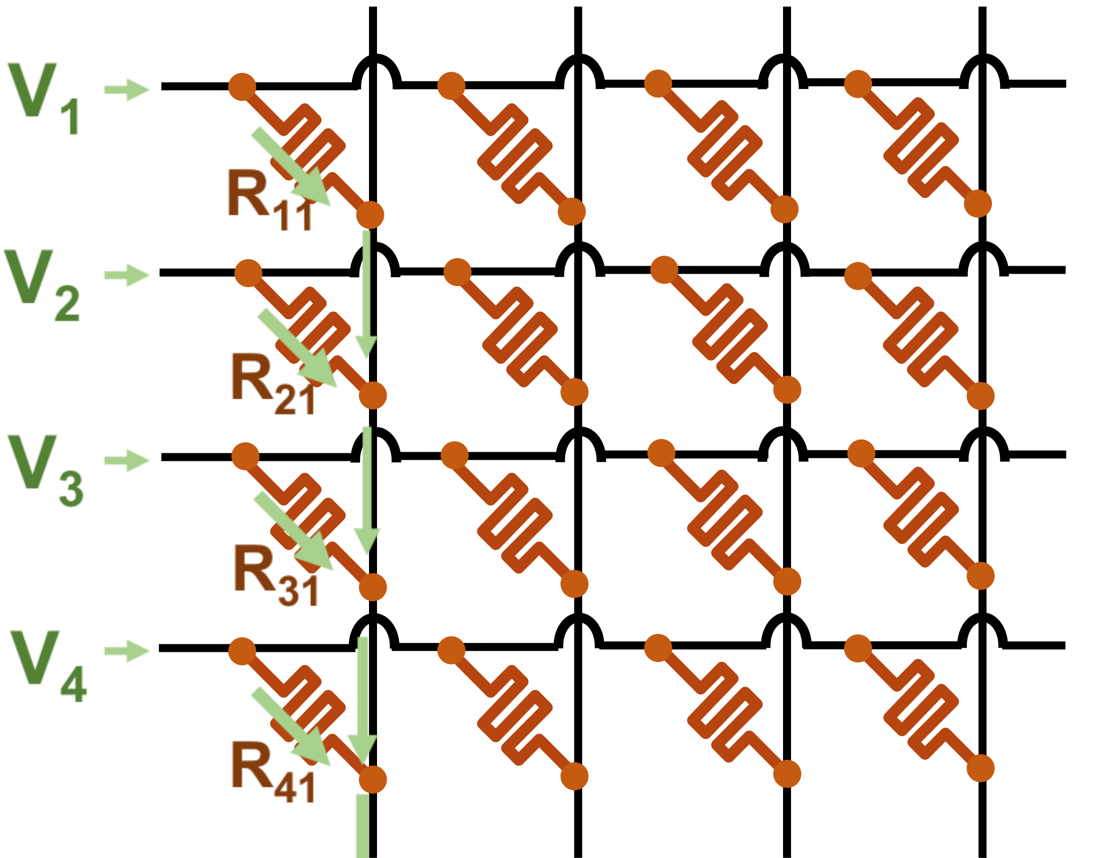
Vector-Matrix Multiplication (VMM)

→ Memristor-based crossbars support VMM

Data movement between memory and accelerator (e.g., GPU or TPU)

→ Computation in Memory (CIM) minimizes data movement

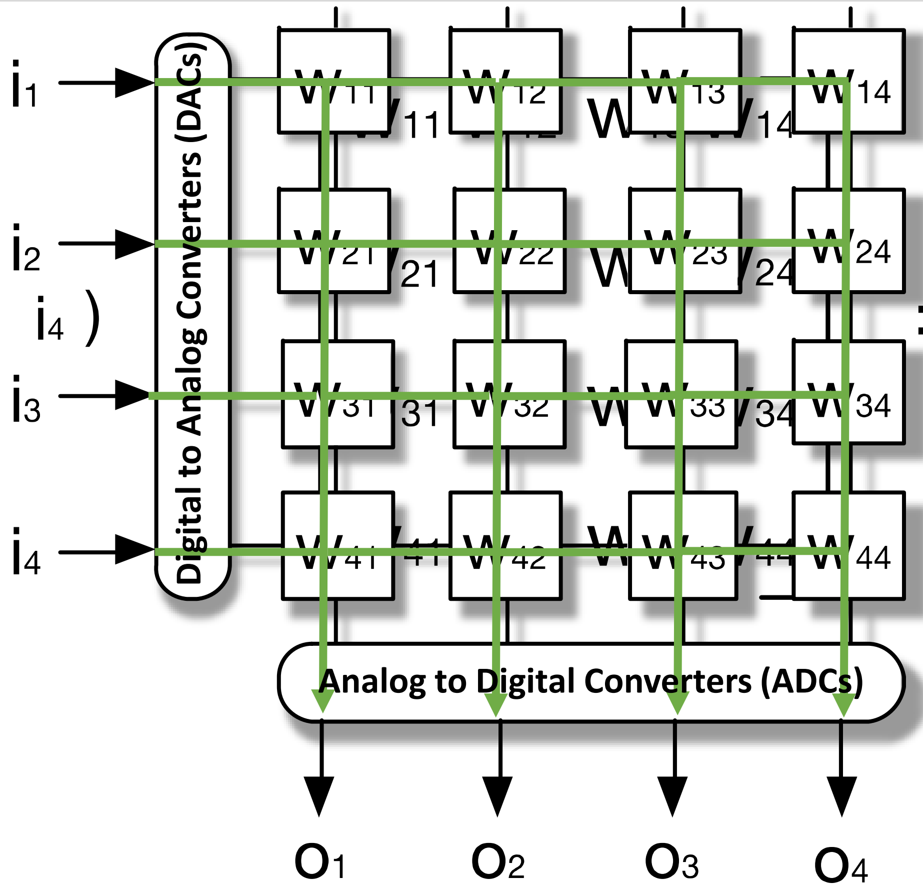
Memristor-based Crossbars



$$I_1 = \frac{1}{R_{11}}V_1 + \frac{1}{R_{21}}V_2 + \frac{1}{R_{31}}V_3 + \frac{1}{R_{41}}V_4$$

VMM in Memristor-based Crossbars

$\begin{pmatrix} i_1 & i_2 & i_3 & i_4 \end{pmatrix}$



$= \begin{pmatrix} o_1 & o_2 & o_3 & o_4 \end{pmatrix}$

VMM in Memristor-based Crossbars

VMM in Accelerators

In Accelerators

$$\begin{pmatrix} i_1 & i_2 & i_3 & i_4 \end{pmatrix} \times \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix} = \begin{pmatrix} O_1 & O_2 & O_3 & O_4 \end{pmatrix}$$

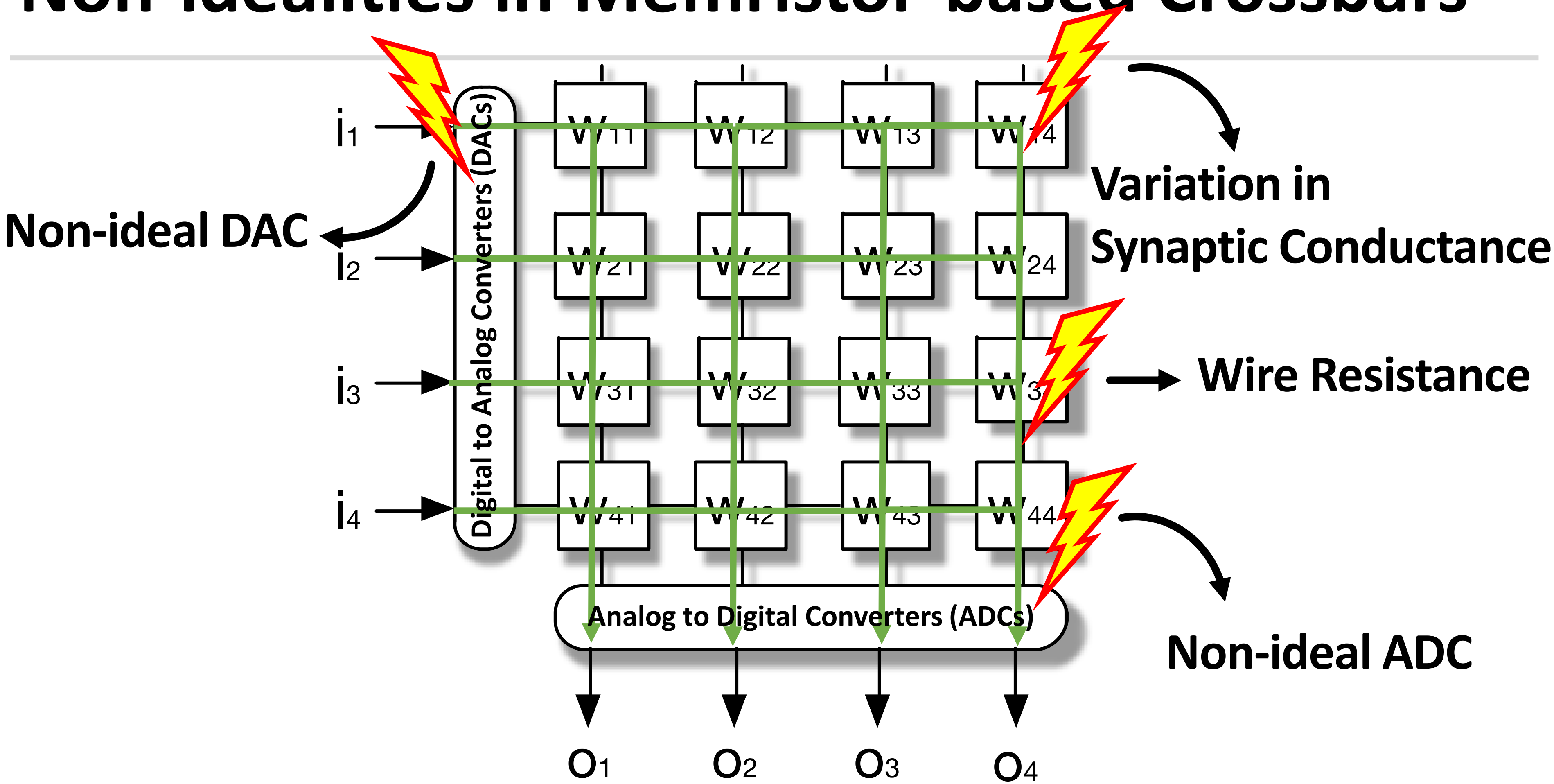
Accurate

VMM in Memristor-based Crossbars

In Memory

$$\begin{pmatrix} i_1 & i_2 & i_3 & i_4 \end{pmatrix} \times \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix} = \begin{pmatrix} O_1 & O_2 & O_3 & O_4 \end{pmatrix}$$

Non-idealities in Memristor-based Crossbars



Non-idealities are **everywhere**

VMM in Memristor-based Crossbars

VMM in Memristor-based Crossbars

In Memory

$$\begin{pmatrix} i_1 & i_2 & i_3 & i_4 \end{pmatrix} \times \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix} = \begin{pmatrix} O_1 & O_2 & O_3 & O_4 \end{pmatrix}$$

VMM in Memristor-based Crossbars

VMM in Ideal Memristor-based Crossbars

In Memory

$$\begin{pmatrix} i_1 & i_2 & i_3 & i_4 \end{pmatrix} \times \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix} = \begin{pmatrix} O_1 & O_2 & O_3 & O_4 \end{pmatrix}$$

Accurate

VMM in Memristor-based Crossbars

VMM in Ideal Memristor-based Crossbars

In Memory

$$\begin{pmatrix} i_1 & i_2 & i_3 & i_4 \end{pmatrix} \times \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix} = \begin{pmatrix} O_1 & O_2 & O_3 & O_4 \end{pmatrix}$$

Accurate

VMM in Real Memristor-based Crossbars

Memory

$$\begin{pmatrix} i_1 & i_2 & i_3 & i_4 \end{pmatrix} \times \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix} = \begin{pmatrix} O_1 & O_2 & O_3 & O_4 \end{pmatrix}$$

Inaccurate

Our Goal

To **realistically evaluate** end-to-end basecalling **accuracy** and **throughput** for memristor-based CIM

Key Idea

To account for the **non-idealities** in **device**, **circuit** and **architecture** of memristor-based CIM and the **overhead** of non-idealities **mitigation techniques**

Outline

Background & Motivation

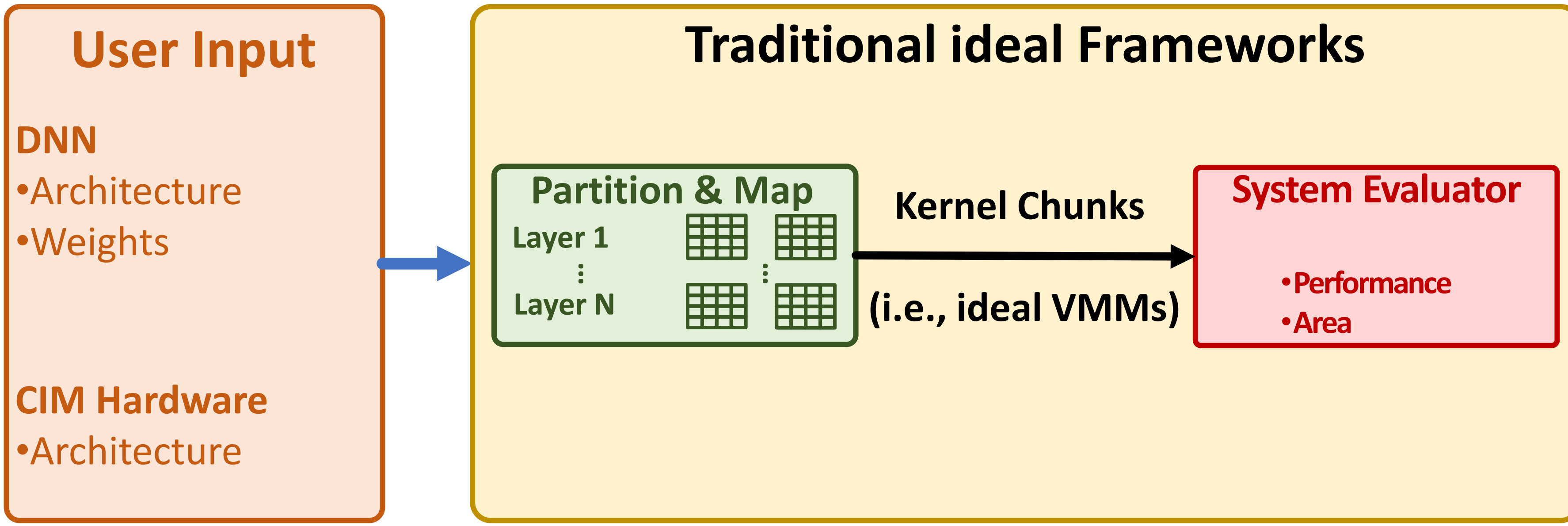
Swordfish: Design & Implementation

Evaluation & Key Results

Takeaways & Summary

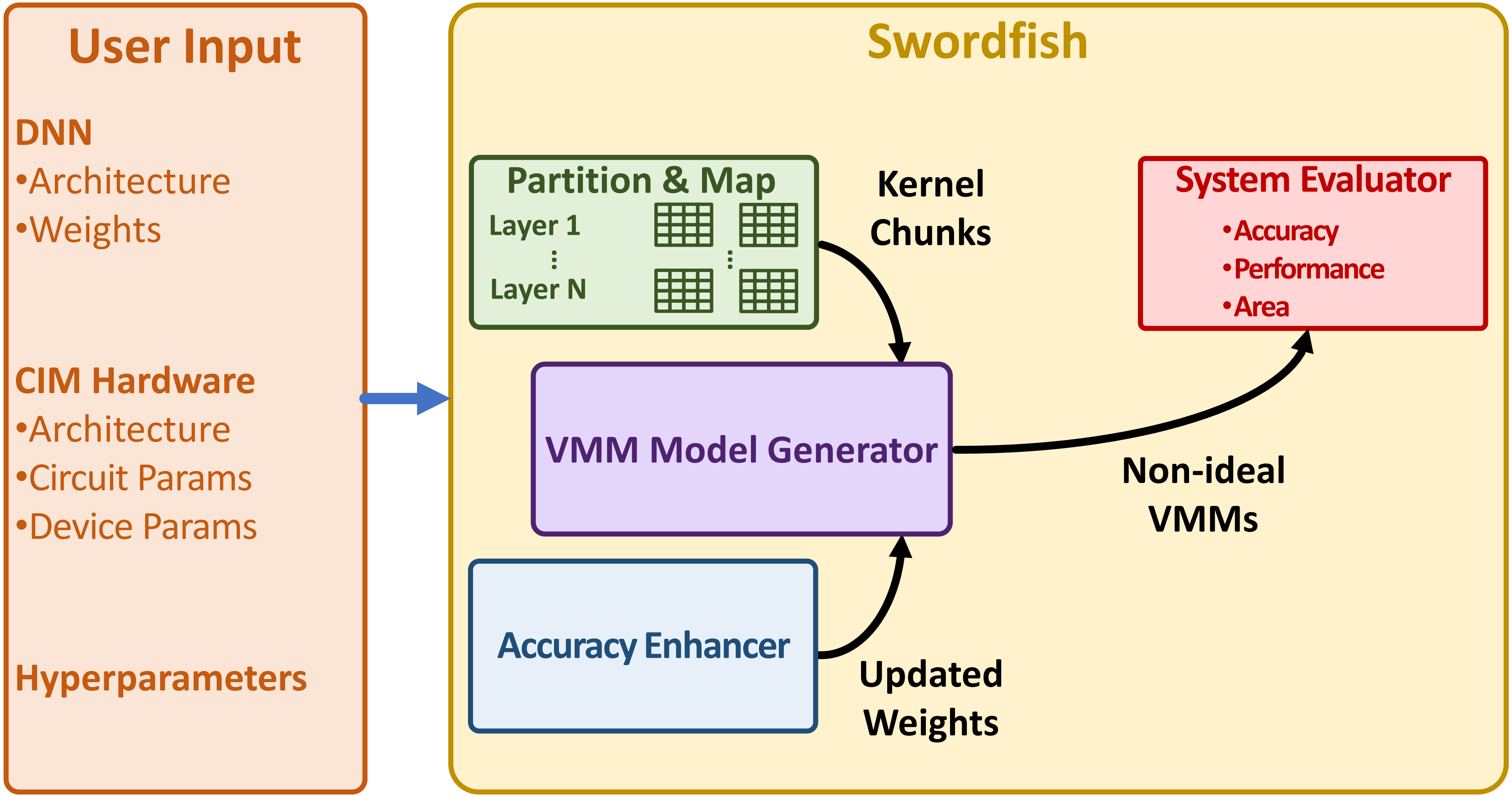
Swordfish vs Other Frameworks

Ideal Memristor-based CIM Frameworks for DNNs



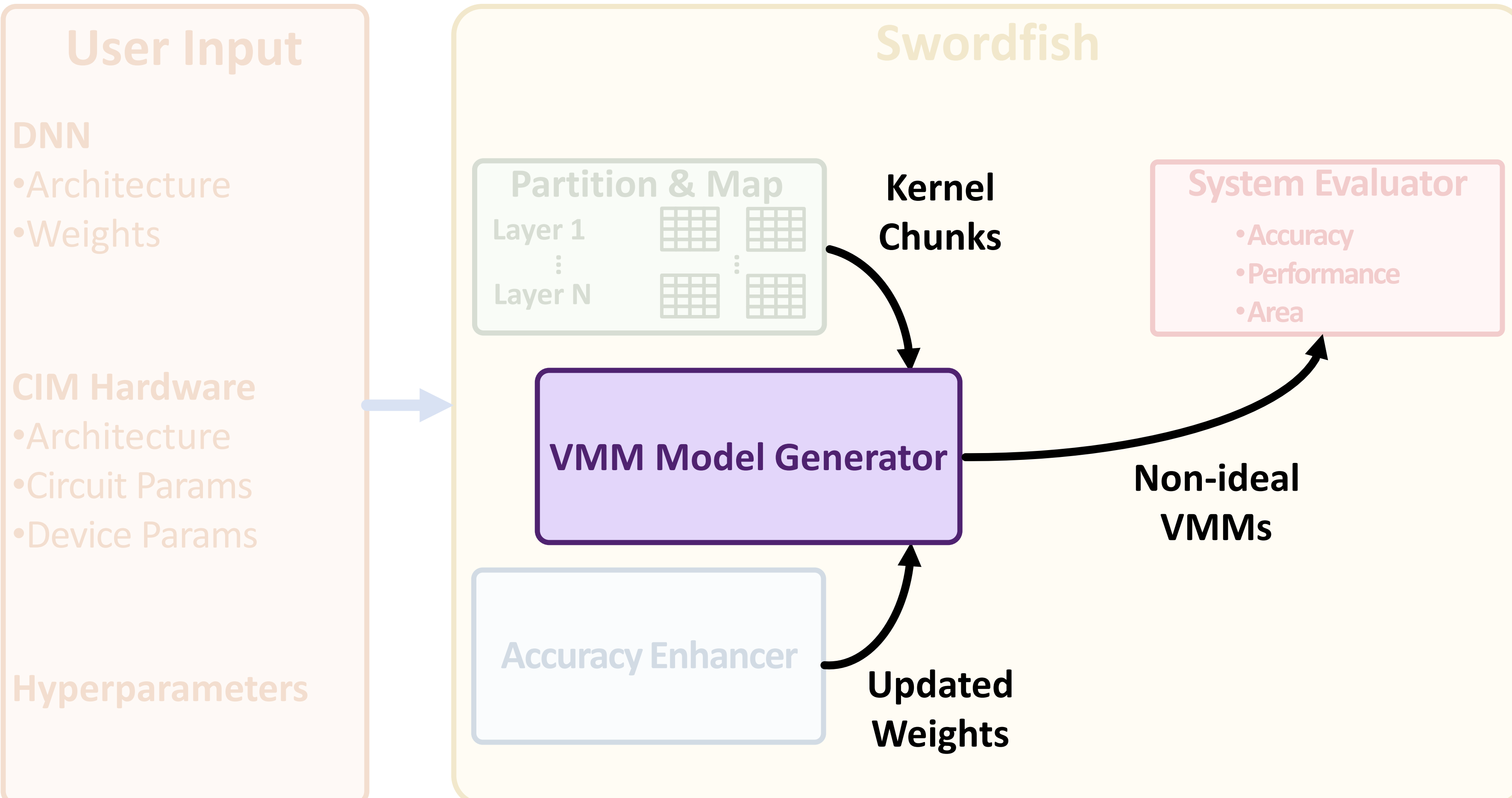
Swordfish Framework - Overview

Realistic Memristor-based CIM Frameworks for DNNs



Swordfish Framework - Overview

Realistic Memristor-based CIM Frameworks for DNNs



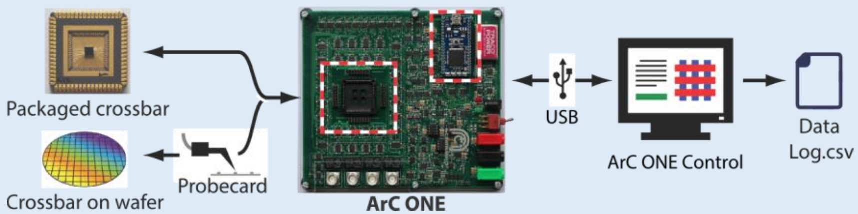
VMM Model Generator

Goal: Capture real output of VMM in presence of non-idealities

Swordfish supports two approaches:

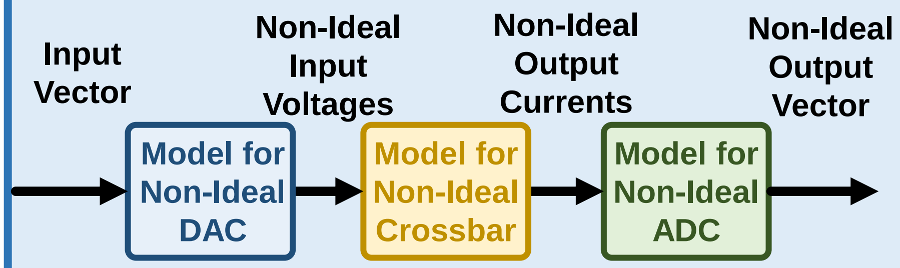
Approach 1:

Real chip **measurement** and **characterizations**



Approach 2:

Analytical Modeling using component's models



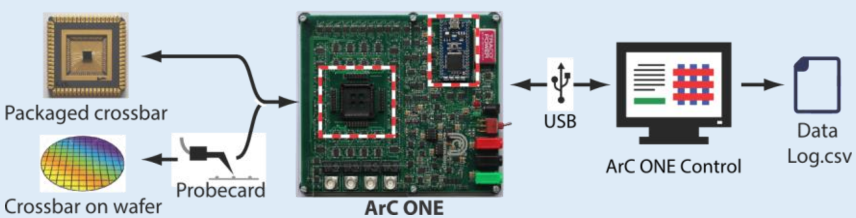
VMM Model Generator

Goal: Capture real output of VMM in presence of non-idealities

Swordfish supports two approaches:

Approach 1:

Real chip **measurement** and **characterizations**

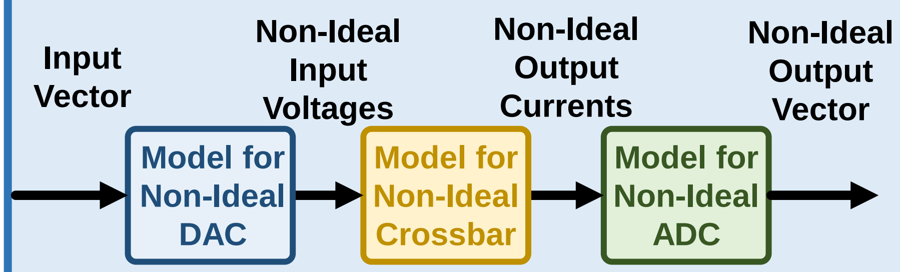


Most **Accurate**

Less **Flexible**

Approach 2:

Analytical Modeling using component's models

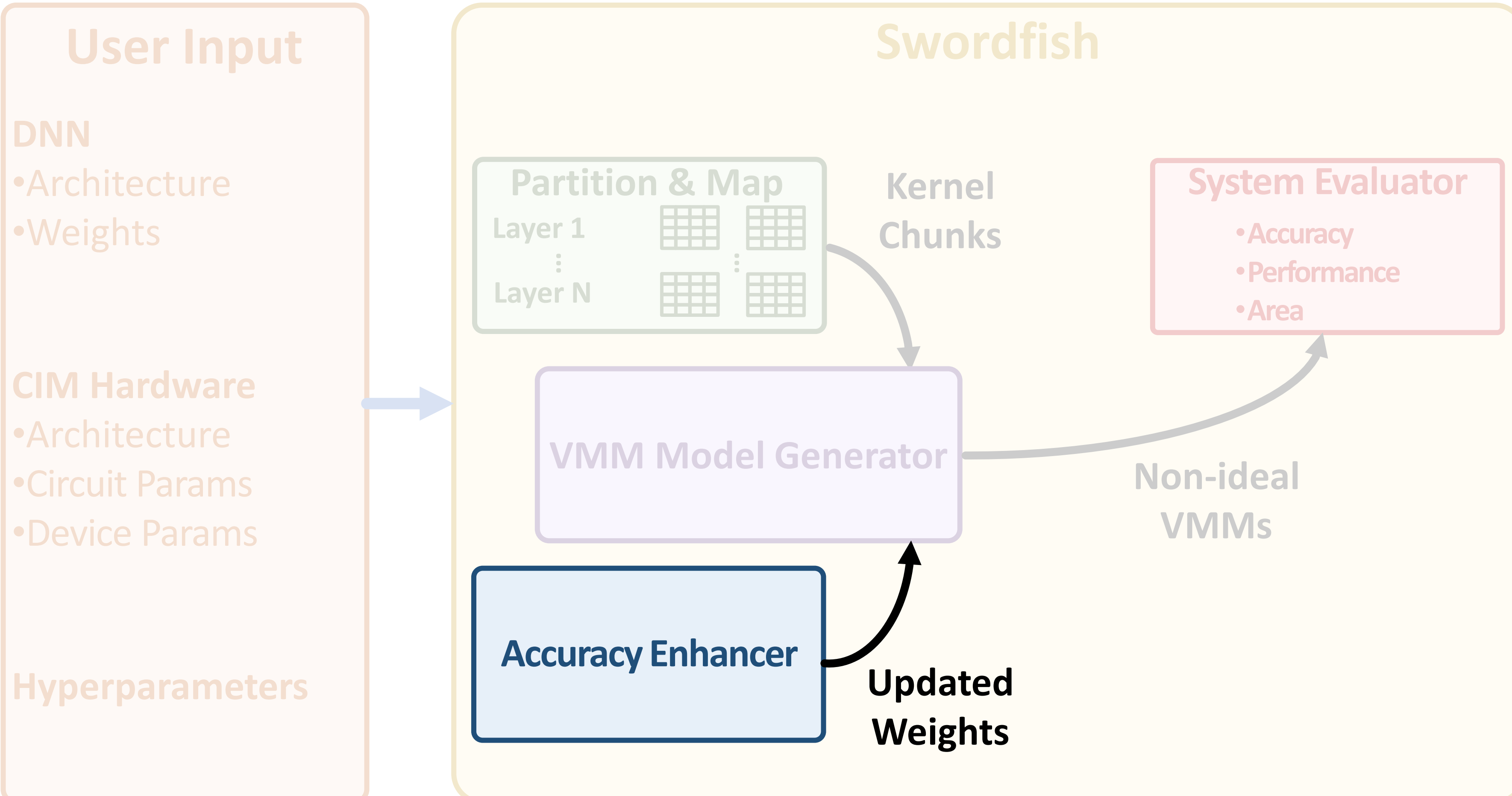


More **Flexible**

Less **Accurate**

Swordfish Framework - Overview

Realistic Memristor-based CIM Frameworks for DNNs



Accuracy Enhancement

Goal: Enhance the accuracy of a VMM by adapting input currents and resistance of memristors based on non-idealities

Swordfish supports four techniques:

1. Analytical Variation Aware Training (VAT)

2. Knowledge Distillation-based (KD) VAT

3. Read-Verify-Write (R-V-W) Training

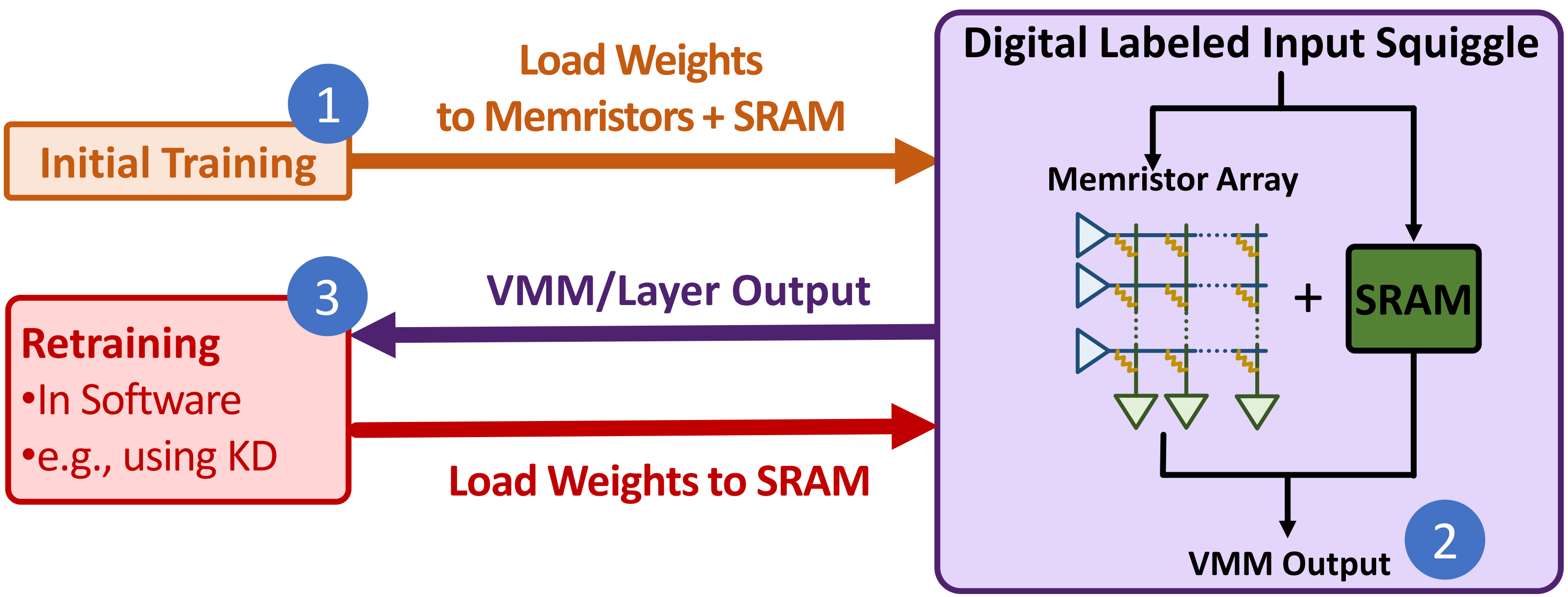
4. Random Sparse Adaptation (RSA) Training

Accuracy Enhancement via Random Sparse Adaptation

Key idea? Map the weights that otherwise would map to error-prone memristor devices to reliable SRAM cells.

RSA in 3 Steps:

1. Initial Training (one-time, on GPU) and distribution of weights
2. VMM operation using both memories
3. Retraining all weights and reload those on **SRAM (only)**



Outline

Background & Motivation

Swordfish: Design & Implementation

Evaluation & Key Results

Takeaways & Summary

Evaluation Methodology: Experimental Setup

- **Authors evaluated**
  - **Basecaller:** Bonito [Oxford Nanopore 2023]
  - **CIM Architecture:** PUMA [Ankit+, ASPLOS 2019]

- **Infrastructure**
  - 2x AMD EPYC 7742 CPU with 500 GB DDR4 DRAM
  - 8x NVIDIA V100

- **Datasets and Workloads** [Wick+ 2019, Zook+ 2019, CADDE 2020]
  - 4 real read and reference genomes with various genome size (D1, D2, D3, and D4)

Evaluated Non-idealities & Enhancement techniques

- **Non-idealities**

1. Synaptic+Wires
2. Sensing+ADC Circuitry
3. DAC+Driver Circuitry
4. Combined

Analytical Models (Approach 2)

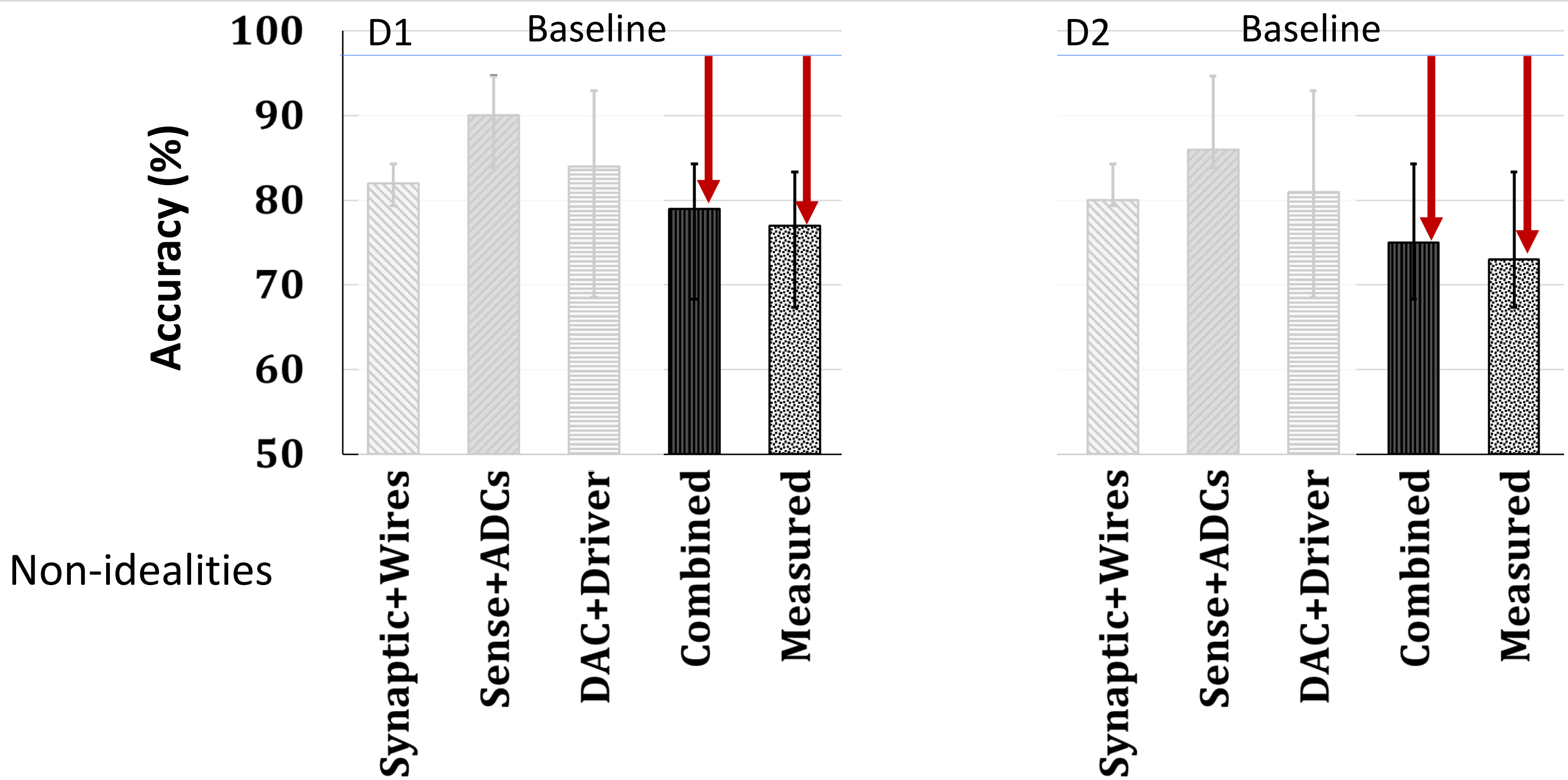
5. Measured

(Approach 1)

- **Accuracy Enhancement Techniques**

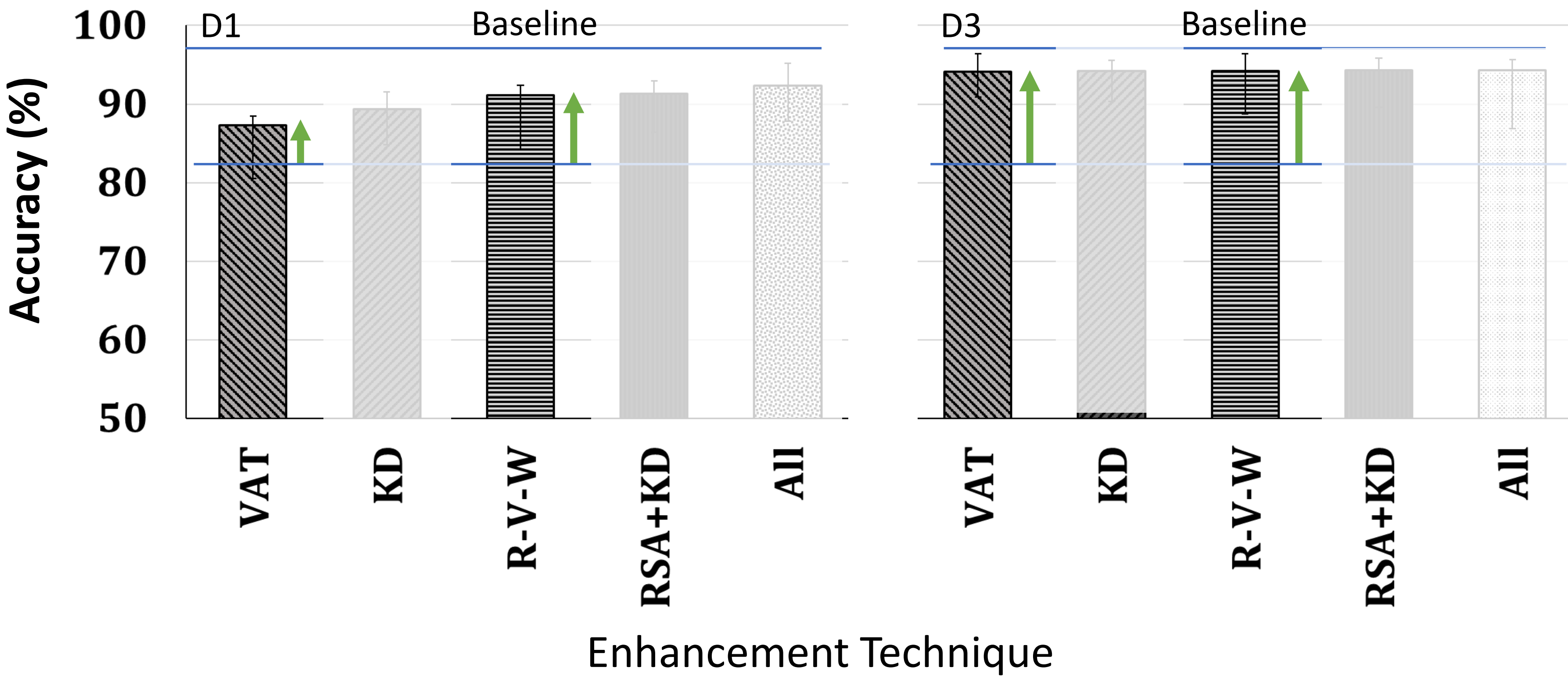
1. Variation Aware Training (VAT)
2. Knowledge Distillation (KD)
3. Read-Verify-Write (R-V-W)
4. Random Sparse Adaptation (RSA) + KD
5. All

Accuracy: All Non-idealities without Mitigation



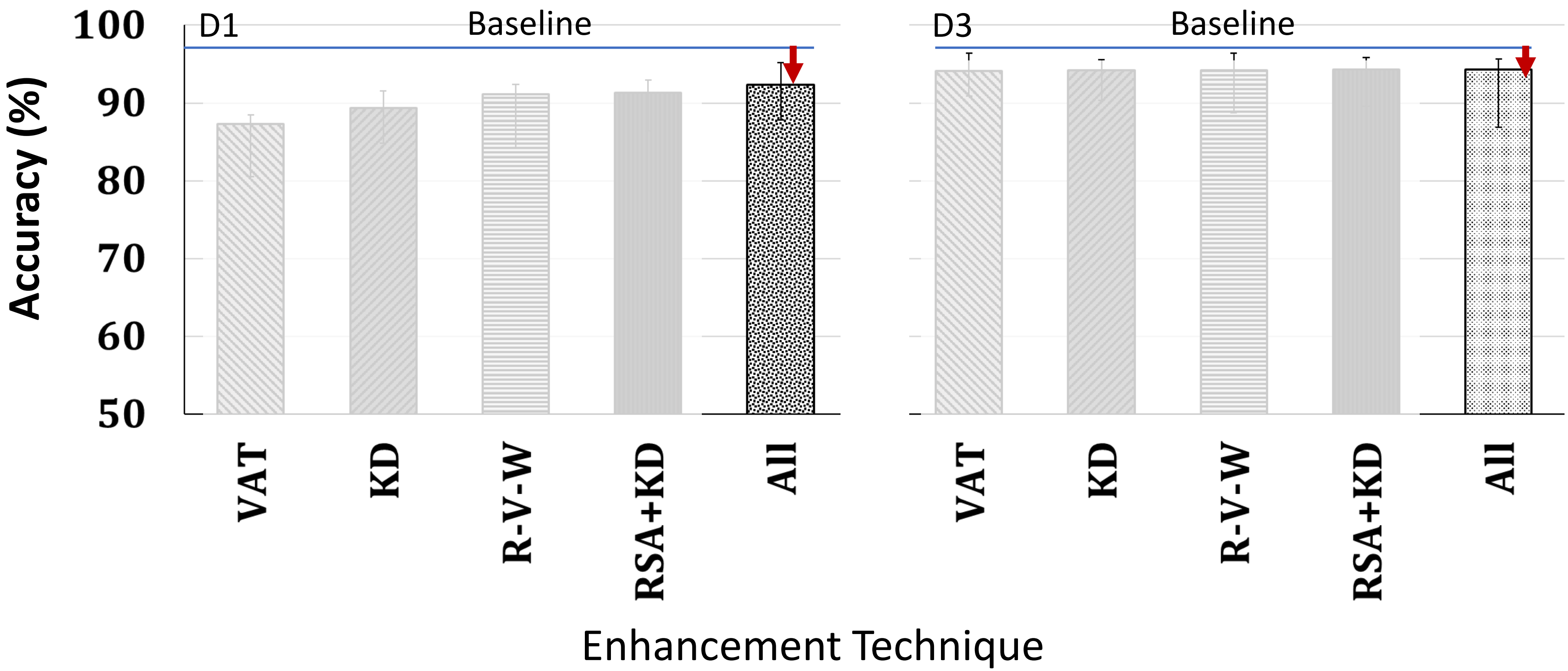
Combined non-idealities leads to significant accuracy loss (**>18%**)

Accuracy: Enhancement Techniques on All Non-idealities



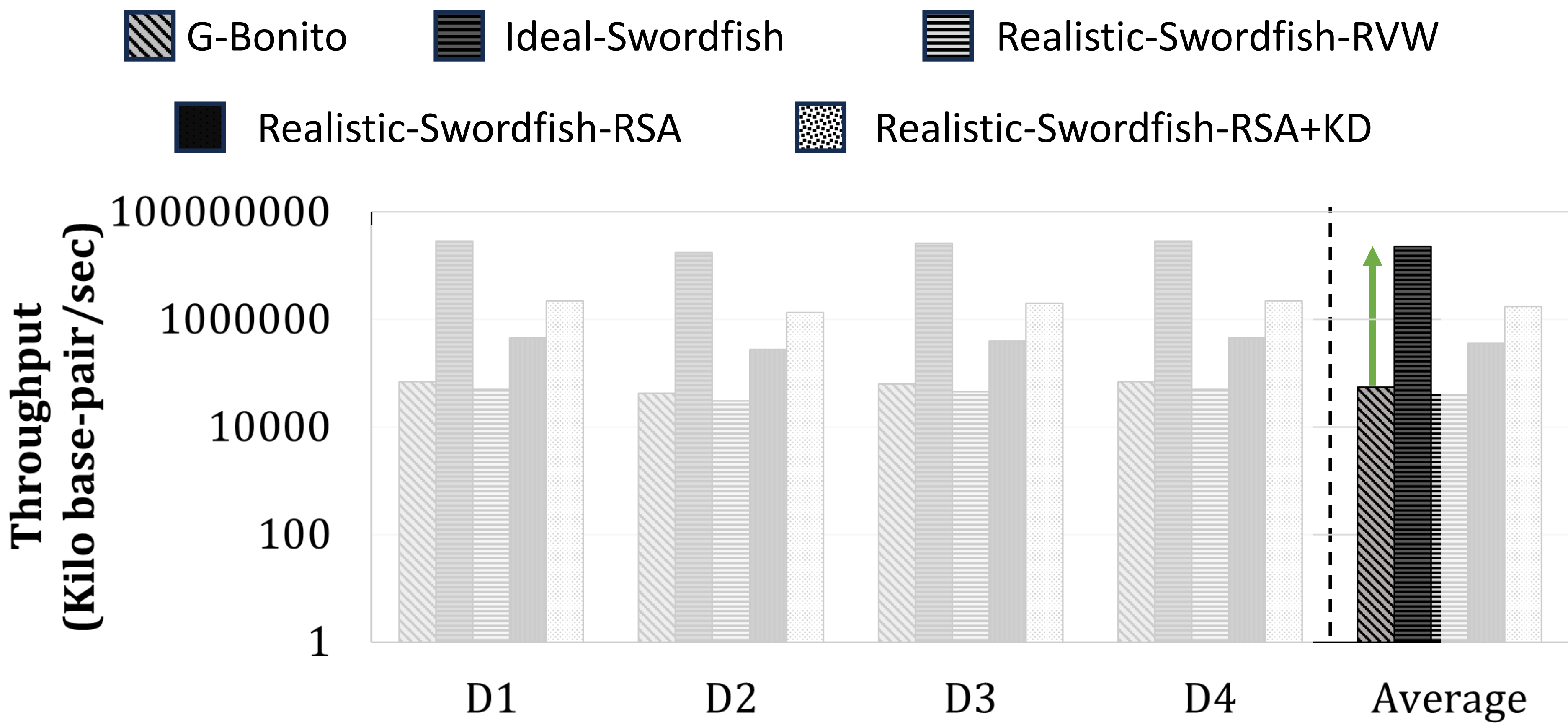
Accuracy enhancement techniques **mitigate** non-idealities,
But differently.

Accuracy: Enhancement Techniques on All Non-idealities



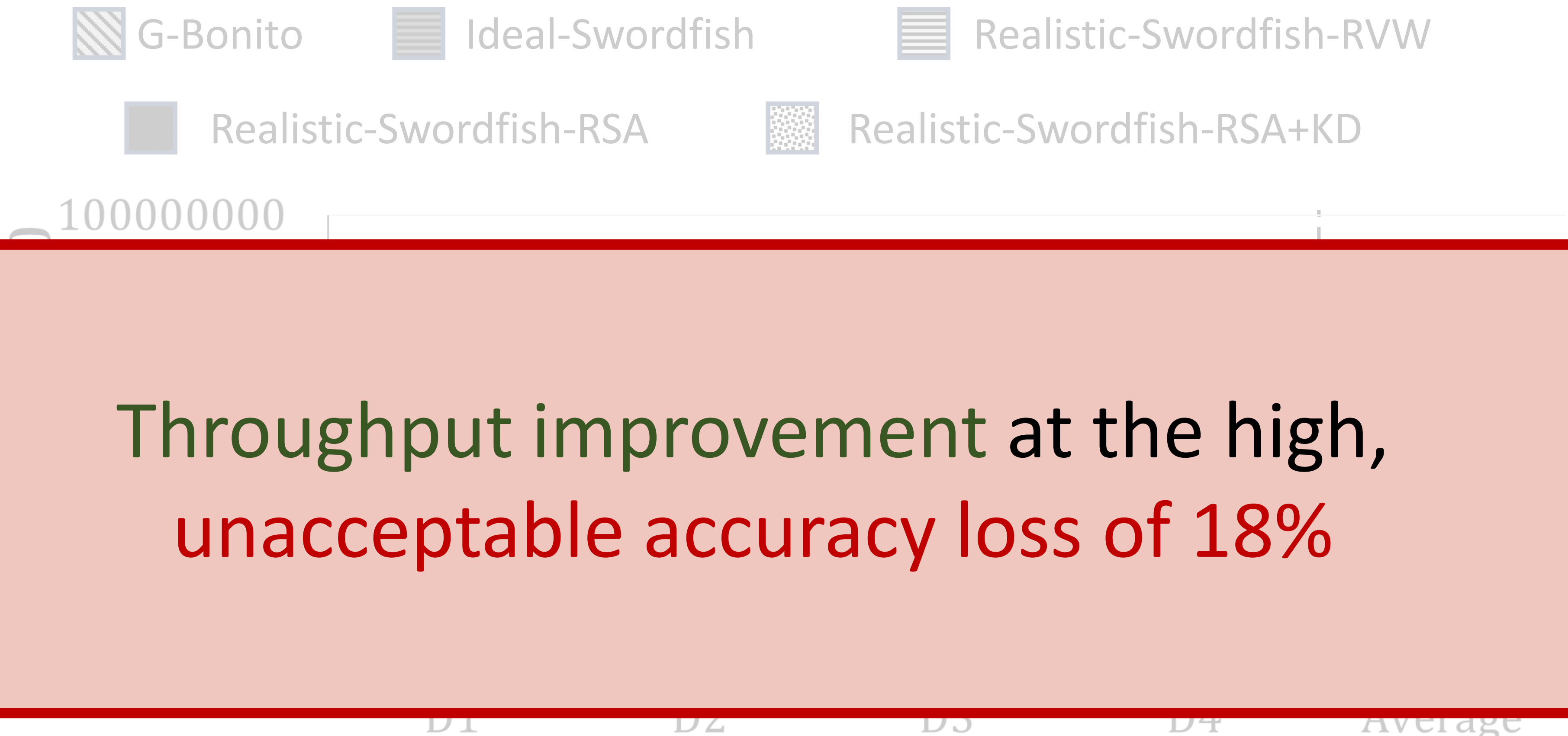
Considerable **accuracy loss** (**>6%**) **even with All** enhancement techniques.

Throughput Analysis



Ideal CIM implementation improves the basecalling throughput over Bonito-GPU by **413.6× on average**

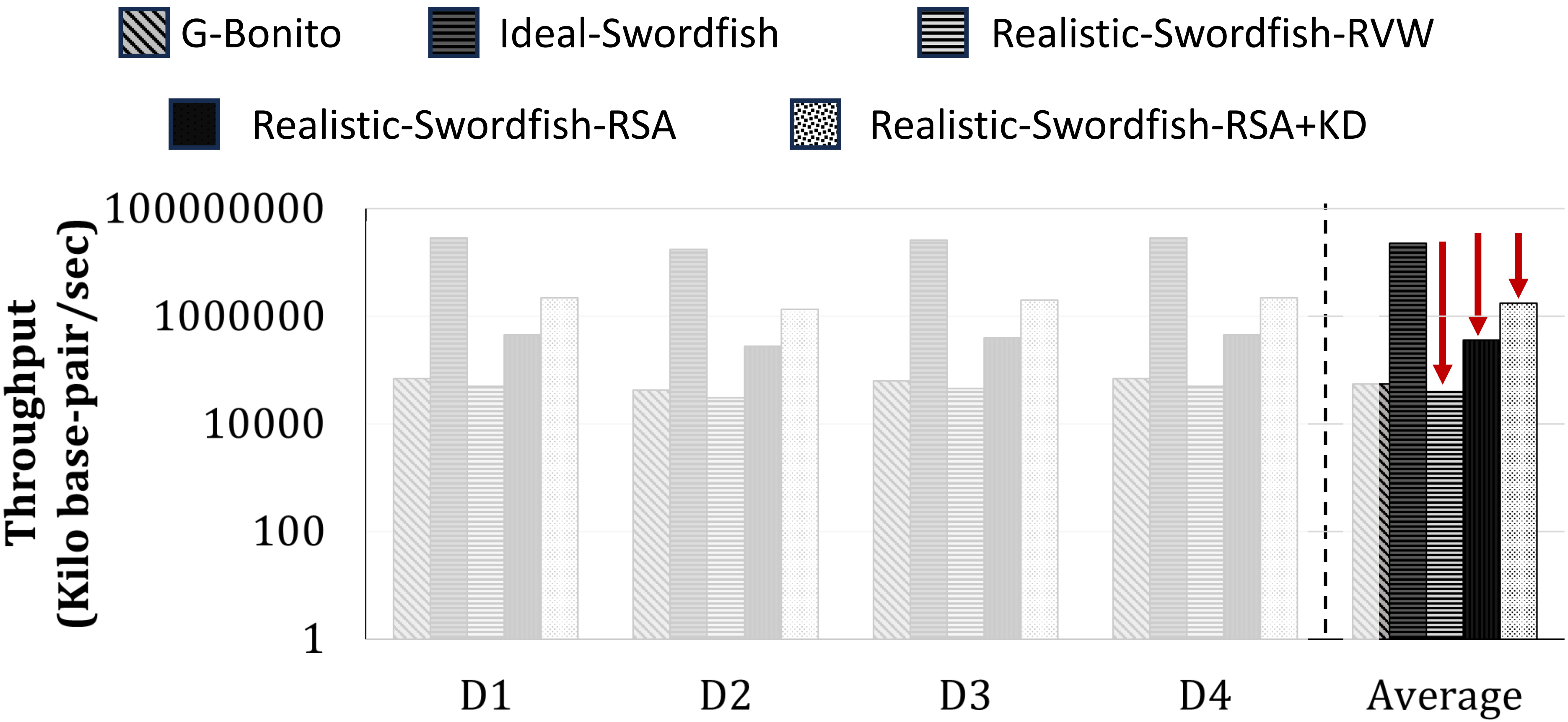
Throughput Analysis



Throughput improvement at the high, unacceptable accuracy loss of 18%

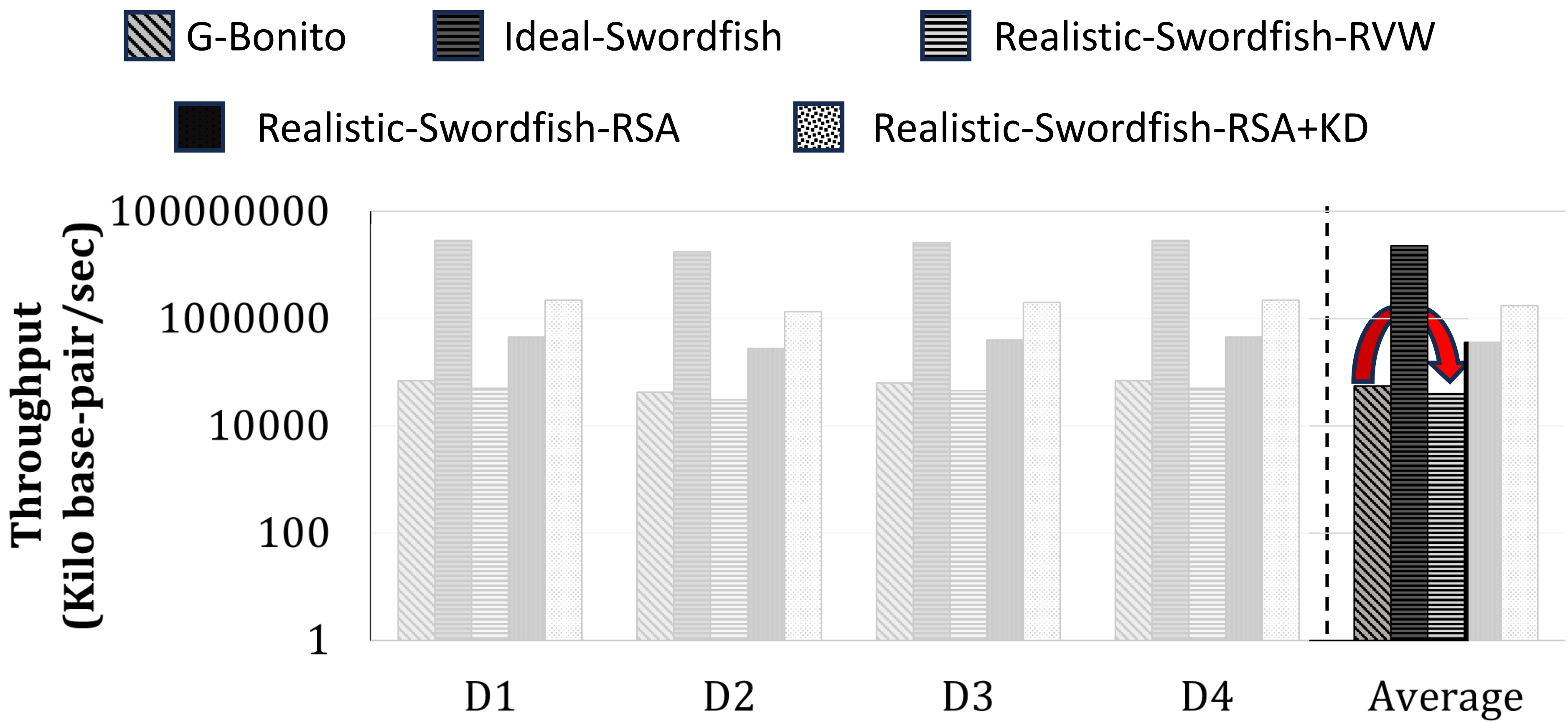
Ideal CIM implementation improves the basecalling throughput over Bonito-GPU by **413.6× on average**

Throughput Analysis



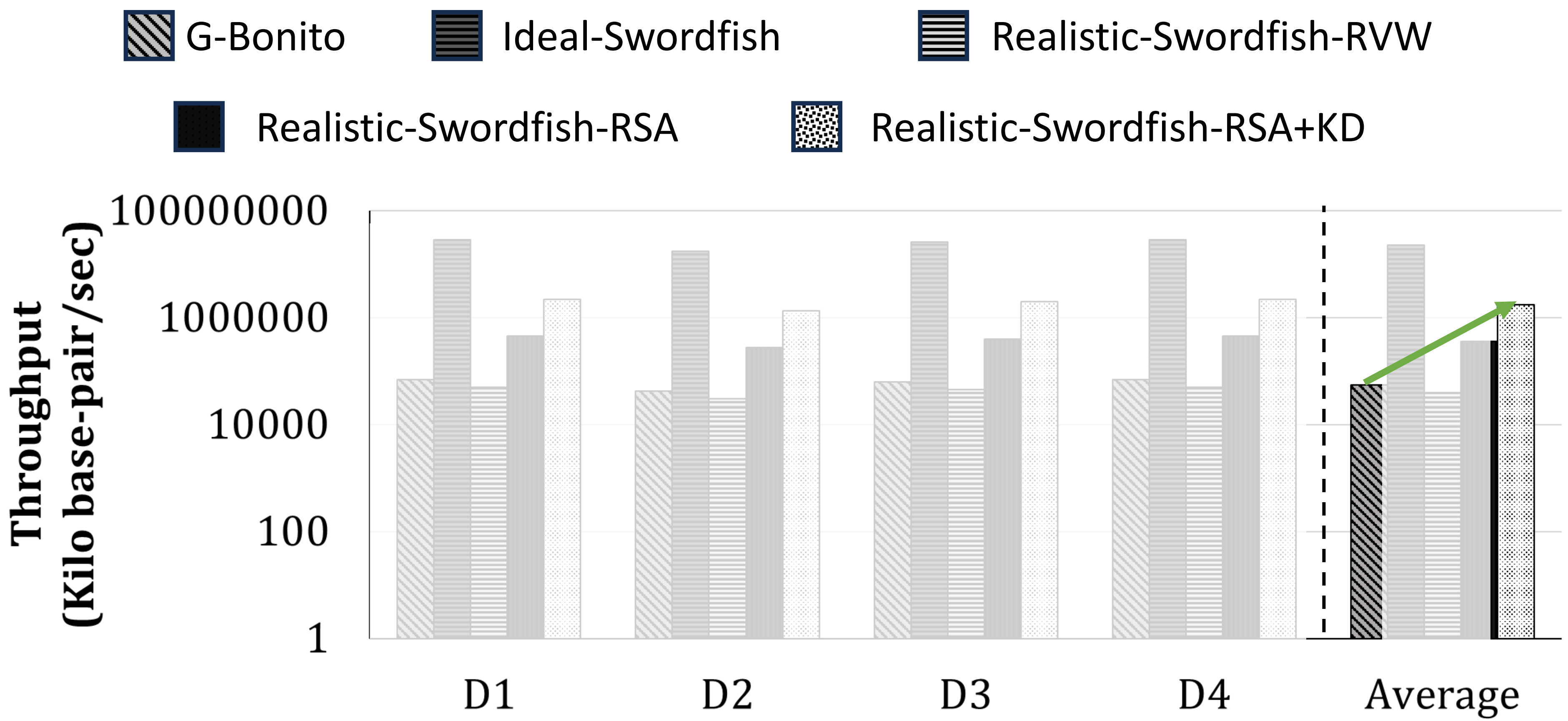
Realistic CIM designs **significantly underperform** ideal design

Throughput Analysis



Some **realistic CIM designs** **degrade** throughput compared to Bonito-GPU

Throughput Analysis



Realistic CIM design using RSA+KD provides on average **25.7× higher throughput** compared to Bonito-GPU

Outline

Background & Motivation

Swordfish: Design & Implementation

Evaluation & Key Results

Improvements & Takeaways

Opportunities for Improvement

- Explore variety of DNN applications
- Overhead introduced by Swordfish framework
- Power and execution analysis for training and inference

Takeaways

The target application for memristor-based CIM **matters**

Swordfish enables **realistic** evaluation of accuracy and performance for DNN-based applications on memristor-based CIM

Non-idealities are **detrimental** to both **accuracy** and **performance**

HW/SW co-designed techniques mitigate inaccuracy the most

Swordfish:

A Framework for Evaluating Deep Neural Network-based Basecalling using Computation-in-Memory with Non-Ideal Memristors

Questions?

Quantum & Computer Engineering

SAFARI
SAFARI Research Group
safari.ethz.ch

TU Delft
Delft University of Technology

AMD

ETH zürich

PUMA Original Architecture

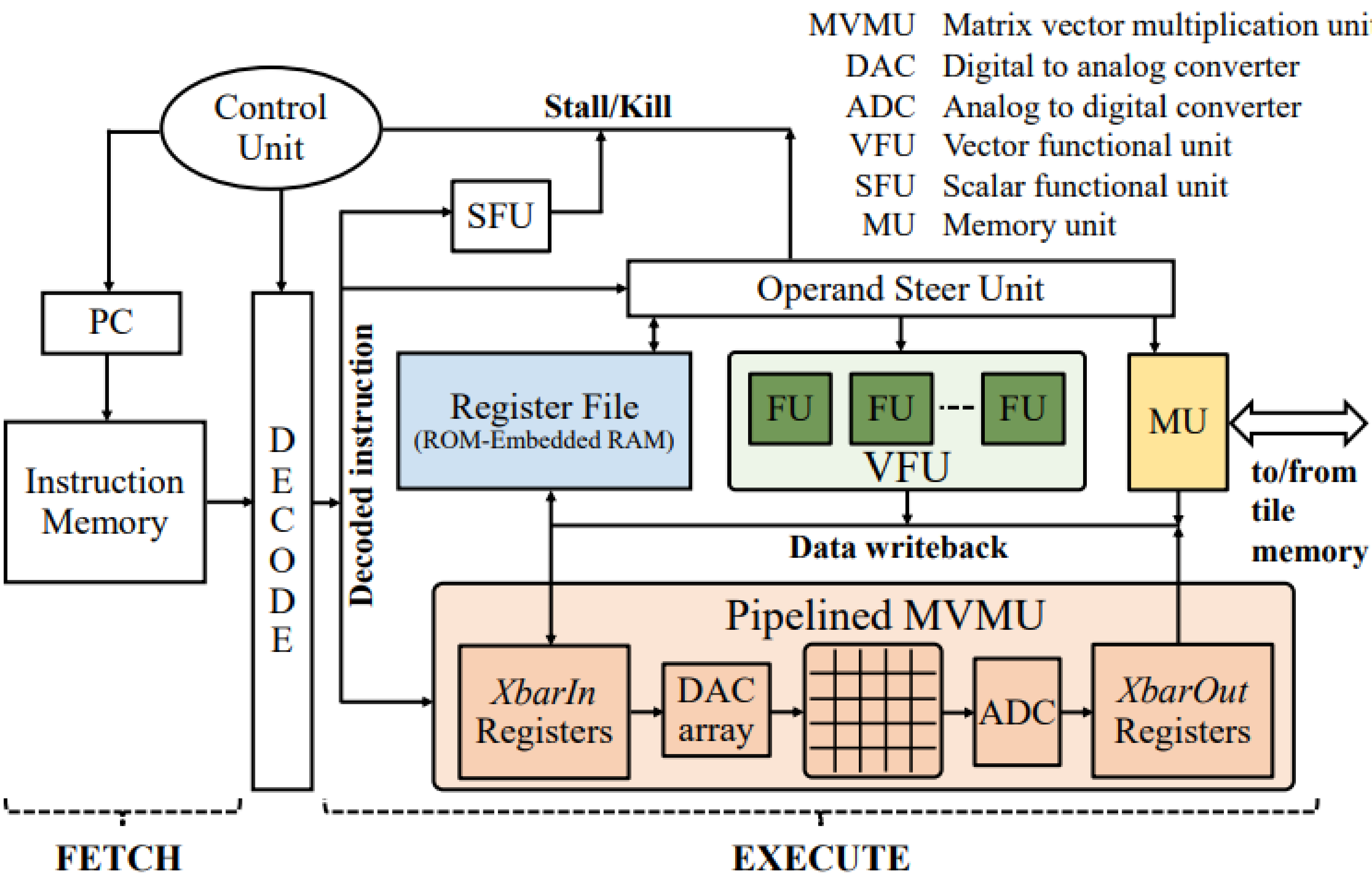


Figure 1. Core Architecture