Noise Aware Finetuning for Analog Non-Linear Dot Product Engine

Anonymous Author(s)
Affiliation
Address
email

Abstract

As interest in analog acceleration for deep neural networks (DNNs) grows, ReRAM-based Dot-Product Engines (DPEs) offer an energy-efficient solution for performing vector-matrix multiplications (VMMs) in the analog domain. However, DPEs require Analog-to-Digital Converters (ADCs), which contribute significantly to area and power overhead, and rely on digital logic for operations such as non-linear activations. This work presents an ADC-less DNN accelerator that leverages Analog Content Addressable Memory (ACAM) to replace ADCs and digital activation units. By training decision trees to approximate activation functions and programming them to ACAMs, the novel Non-linear DPE (NL-DPE) enables arbitrary activation to be implemented directly in the analog domain, supporting a broader range of future DNN architectures. Additionally, we explore the inherent noise present in real devices of both DPE and ACAM and propose noise-aware finetuning techniques that mitigate accuracy loss, demonstrating notable improvements.

1 Introduction

DNNs have significantly grown in size and complexity, driving an increasing demand for memory bandwidth and computation. As DNN models continue to expand, the energy consumption associated with frequent data movement between memory and processing units has become a significant bottleneck, known as the *memory wall*. Such bottlenecks have led to the development of In-Memory Computing (IMC) accelerators, where computation occurs directly within the memory. Among the various IMC approaches, ReRAM-based analog computing stands out as one of the most promising solutions due to its potential for high energy efficiency and scalability. ReRAM cells can be organized in a crossbar structure to design DPEs, which perform VMMs in the analog domain in a single step, achieving low energy consumption and high parallelism [Shafiee et al. (2016); Ankit et al. (2019)].

However, DNNs consist of more than just VMMs; they also require non-linear activations, which are typically executed using digital logic. Therefore, DPE's analog outputs must be converted into digital signals via ADCs. Unfortunately, ADCs are both energy and area-inefficient, significantly impacting the overall efficiency of ReRAM-based accelerators. ADCs can consume more than 30% of the chip area and account for over 50% of the total power consumption, creating a substantial bottleneck in achieving truly energy-efficient DNN accelerators [Shafiee et al. (2016); Ankit et al. (2019)].

Another challenge for ReRAM-based DNN accelerators is the inherent noise in analog computing. While many existing works tend to overlook or underestimate its effect, our experiments demonstrate that noise plays a crucial role in determining the accuracy of computations on real hardware. In fact, without careful attention to noise, accuracy can degrade to unacceptable levels, as noted also in other studies [Mao et al. (2022)]. Although some efforts have been made to address this issue [Momeni et al. (2024)], novel computation primitives may require to carefully handle new sources of noise.

Submitted to the Second Workshop on Machine Learning with New Compute Paradigms at NeurIPS (MLNCP 2024). Do not distribute.

In this work, we propose the NL-DPE, an analog DNN accelerator design that eliminates the need for ADCs and digital activation units. We convert activation functions into decision trees, which are then mapped onto the programmable ACAM for analog computation. Since ACAM accepts analog inputs and produces digital outputs, ADCs are no longer required after the DPE. To address the reliability issues of analog computation, we measure noise directly from real devices and develop detailed noise models that account for various sources, including programming inaccuracies and conductance fluctuations, etc. Noise models are integrated into a fine-tuning process to minimize the accuracy gap caused by noise.

2 Analog Computing with ReRAM

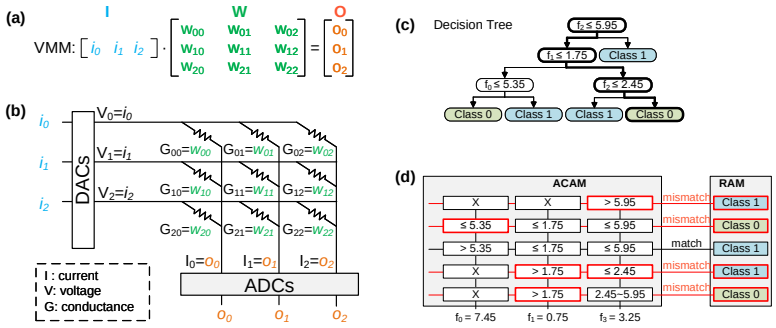


Figure 1: **(a)** An example of VMM computing. **(b)** Computing the VMM in DPE. **(c)** An example of a decision tree. **(d)** Mapping the decision tree onto ACAM and performing an inference.

Fig. 1(a) shows an example VMM operation ($\mathbf{O} = \mathbf{I} \cdot \mathbf{W}$). $\mathbf{O} \in \mathbb{R}^3$, $\mathbf{I} \in \mathbb{R}^3$ and $\mathbf{W} \in \mathbb{R}^{3\times3}$. Fig. 1(b) illustrates computing the VMM operation in the DPE with a 3×3 crossbar. A ReRAM cell is placed at every intersection of the horizontal wires and vertical wires. Weight elements (W_{ij}) are programmed as ReRAM conductance (G_{ij}), and input elements(i_i) are represented by voltage (V_i) on horizontal wires. Following Kirchhoff's law and Ohm's law, the current (I) from each vertical wire conveys the dot-product result of the input vector and the weight matrix. The analog VMM operation is completed in a single step within the DPE, achieving high computing parallelism.

Recently, ReRAM has been used to build Analog Content Addressable Memories (ACAMs), accelerating tree-based machine learning algorithms in the analog domain [Li et al. (2020); Pedretti et al. (2021b, 2023)]. Unlike digital CAM, which is limited to comparing a single input bit with a stored bit, an ACAM cell can compare an analog input value against a stored analog range. To map a trained decision tree of Fig. 1(c) onto an ACAM array, we traverse each leaf node back to the root, storing the feature thresholds along the path in a row of ACAM cells (Fig. 1(b)). Wildcard cells ('X') indicates that the particular feature is irrelevant along that path, thus the full range is programmed. Given an input feature vector during inference, if all feature values fall within the ranges stored in a row, the corresponding match line will be activated, retrieving the predicted class from an adjacent RAM. ACAMs have thus analog input, but digital output representing a match of mismatch.

3 Non-Linear Dot Product Engine

Fig. 2 shows a two-layer snippet from a larger neural network mapped to a conventional DPE (a) and the proposed NL-DPE (b). The first layer is followed by a Sigmoid activation function, while the second layer has no activation.Digital-to-Analog Converters (DACs) are required to convert the digital input into analog signals for computing VMM with the weights in the DPE. In conventional DPE-based accelerators (Fig. 2(a)), activations are typically performed in the digital do-

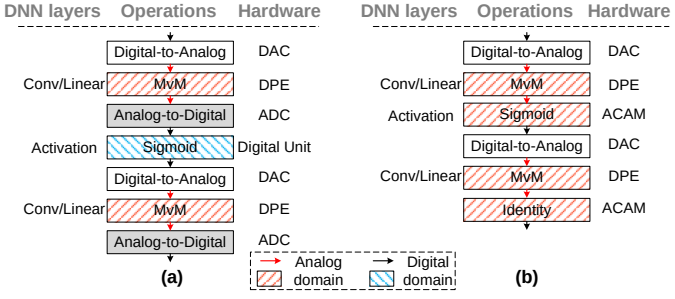


Figure 2: Mapping DNN on **(a)** converntional DPE and **(b)** NL-DPE.

main, thus ADCs are then needed to convert the analog output from the DPE back into digital form. In our proposed NL-DPE (Fig. 2(b)), we use ACAM to compute activations. As ACAM accepts analog input, ADCs are no longer needed after the DPE. The output of our ACAM is a digital signal, DACs are still needed before the next DPE. To replace the ADC, ACAM is programmed as an identity function, a special case of activation.

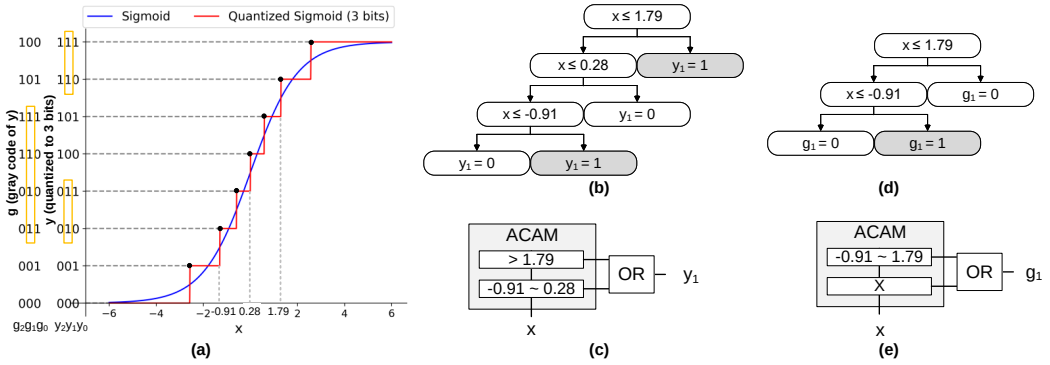


Figure 3: **(a)** Analytical and 3-bit quantized sigmoid function using conventional binary (y) and Grey code (g)format.**(b)** Trained decision tree to predict the second MSB of the activation with conventional binary representation (y_1) and **(c)** its mapping on ACAM. **(d)** Trained decision tree to predict the second MSB of the activation with Grey coding (g_1) and **(e)** its mapping on ACAM.

Fig. 3(a) demonstrates how ACAM is used to compute the Sigmoid activation with a 3-bit output. In this example, we assume 0 and 1 are quantized to 000_b and 111_b respectively, and the remaining 6 values are evenly distributed between 0 and 1, as shown by the y axis in Fig. 3(a). Each output bit (y_2, y_1, and y_0) is computed using a separate decision tree, treating each as a binary classification task. The training dataset contains only one feature (input x), with the output bit as the target. For example, to train the decision tree for the second most significant bit (MSB) (y_1), the target is set to 1 when the input x falls within the range of -0.91 to 0.28, or when x exceeds 1.79, as shown in Fig. 3(a). Fig. 3(b) depicts the trained decision tree and Fig. 3(c) its mapping onto an ACAM array. Unlike traditional decision trees, which generalize to predict unseen inputs, these trees memorize the exact patterns from the training data, forcing overfitting. Note that the output of the ACAM already represents the bit value, there is no need to access to an attached memory as opposed to a Look Up Table (LUT) approach [Zhu et al. (2022)].

Since activations are typically monotonic functions, the less significant bits in the output tend to toggle more frequently between 0 and 1, leading to deeper decision trees. To mitigate this, we propose an encoding scheme based on Gray code to reduce bit toggling. Fig. 3(a) also shows the Gray code for the output y on the left (denoted as g), where only one bit changes between consecutive values. Fig. 3(d) shows the decision tree trained using Gray code as the output format and Fig. 3(e) its mapping to ACAM.

To enable subsequent computations, the Gray code output from the ACAM must be converted back to its original binary form, as illustrated in the Appendix Section A.

4 Mapping DNN layers into NL-DPE

Although ReRAM is inherently analog devices, most previous work maps DNN weights onto DPEs in a digital manner [Shafiee et al. (2016)]. Typically, this involves quantizing the full-precision floating-point DNN weights into fixed-point format (e.g., 8 bits) and using ReRAM cells to store portions of the quantized weights. Fig. 4(a) shows an example with four ReRAM cells used to store an 8-bit weight, with each cell representing 2 bits, an approch called bit slicing [Shafiee et al. (2016)] However, this approach is suboptimal, as it doesn't fully use the analog potential of ReRAM.

In our design, we take advantage of the analog properties of ReRAM to map unquantized DNN weights (Fig. 4 (b)), with a technique often referred to as *analog slicing* [Pedretti et al. (2021c); Song et al. (2024)]. Programming ReRAM cells involves an iterative program-and-verify (PV) process to reach a target conductance. However, achieving an exact value can be time-consuming, so a tolerance

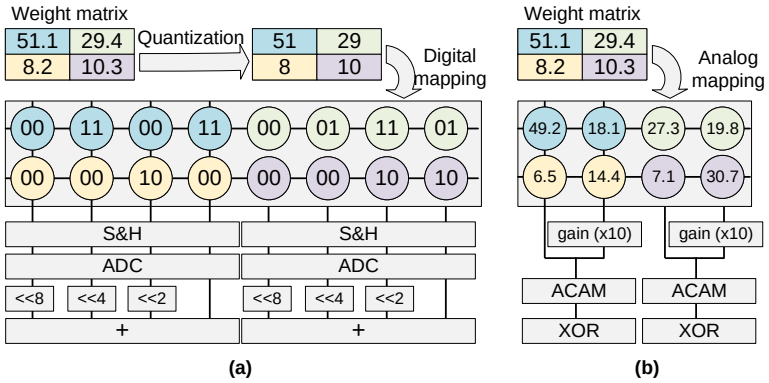


Figure 4: Mapping a weight matrix onto DPE in **(a)** digital form and **(b)** analog form.

threshold δ is used to stop once the error between the programmed conductance (G) and its target (W) is close enough (i.e., $|W - G| < \delta$). Typically, an error ϵ remains ($\epsilon = W - G$). In analog slicing a first Most Significant Cell (MSC) is iteratively programmed. In the example of Fig. 4(b), considering the target weight $W_{00} = 51.1\mu$S the programmed MSC is $G^M_{00} = 49.2\mu$S, resulting in an error $\epsilon^M_{00} = 51.1 - 49.2 = 2.7\mu$S.

After programming all the MSCs, the resulting error is mapped into a second ReRAM cell, namely the least significant cell (LSC). First, the maximum error ϵ^M_{max} is used to compute a *gain* for programming the error exploiting the conductance full-scale range. For example, if the conductance range is 0 to 100 μS and the maximum error is 10μS, a gain $g = 10$ can be considered, as in our example. Thus, the LSC corrects the ϵ^M_{ij} error by being programmed to a conductance that represents the scaled-up error. In our example, the LSC is programmed targeting 27 μS. At the end of program-and-verify algorithm, the actual conductance of the LSC might be $G^L_{00} = 18.1\mu$S, with a programming error $\epsilon^L_{00} = 27 - 18.1 = 8.9\mu$S. During inference, both cells receive the input at the same time, and their output currents are combined, with the LSC's current scaled down to compensate for the error, attenuating it by g^{-1}. Thus, the final programming error for each weight is

$$\begin{aligned} \epsilon^{TOT}_{ij} &= W_{ij} - (G^M_{ij} - g^{-1}G^L_{ij}) \\ &= W_{ij} - (G^M_{ij} + g^{-1}[g(W_{ij} - G^M_{ij}) + \epsilon^L_{ij})] \\ &= W_{ij} - G^M_{ij} - W_{ij} + G^M_{ij} - g^{-1}\epsilon^L_{ij} \\ &= g^{-1}\epsilon^L_{ij} \end{aligned} \tag{1}$$

reducing the conductance error by a factor $g >> 1$. In our example, the final noise is thus $51.1 - (49.2 + 18.1 \div 10) = 0.09\mu$S. We use the programming strategy described in [Mao et al. (2022)] to make the actual conductance always smaller than the target value.

For our experiments, we considered TaOx ReRAM devices integrated in 28 nm CMOS technology [Sheng et al. (2019)] and extracted a detailed noise model-based measurement on a fabricated crossbar test chip. The detailed noise model are described in Appendix Section B.

5 Noise Aware Finetuning for NL-DPE

With noise present in both DPE and ACAM of the NL-DPE, accuracy can be significantly affected. We propose a progressive approach that replaces ADCs and digital activation units with ACAM while accommodating noise. Fig. 5 outlines the process, which starts from a trained DNN model and produces a noise-tolerant model ready for deployment on hardware. In Step 1, activation quantization is applied, as DPE inputs must be processed bit by bit (i.e., input bit slicing [Shafiee et al. (2016)]). Step 2 introduces noise by sampling from the model in Appendix Section B and injecting it into the DPE, followed by a few iterations of noise aware fine-tuning (NAF) to compensate for DPE noise. This finetuning also helps to recover the accuracy lost due to quantization in Step 1. In Step 3, the quantization parameters are exported to generate the training data for the decision trees. Specifically, we need the activation function type to generate the graph similar to Fig. 3(a), and the

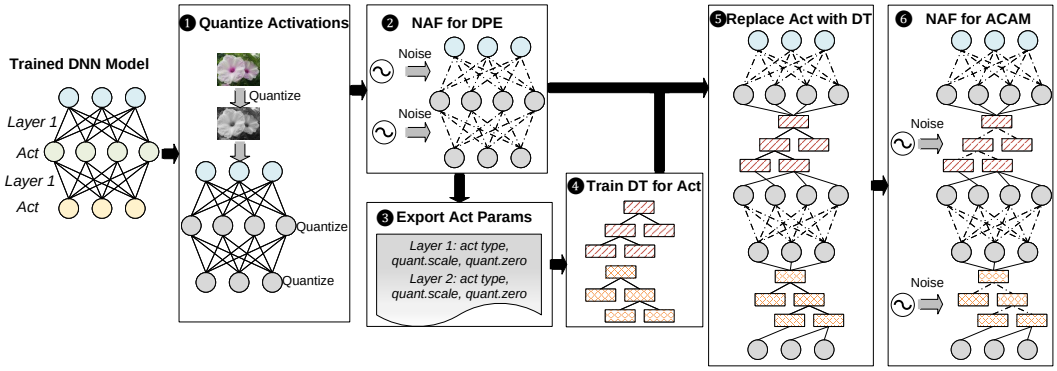


Figure 5: Steps of replacing activations with decision trees and using NAF to tolerate noise in NL-DPE.

quantization parameters (i.e., scaling factor and zero point) that are used to generate the output bit format (equivalent to y in Fig. 3(a)), which are then converted into Gray code format. Step 4 involves training decision trees for each activation using tools like scikit-learn. In Step 5, these decision trees replace the original activations in the DNN. Finally, in Step 6, we sample again from the noise model in Appendix Section B and inject noise into the decision tree thresholds, performing additional NAF iterations to account for ACAM noise.

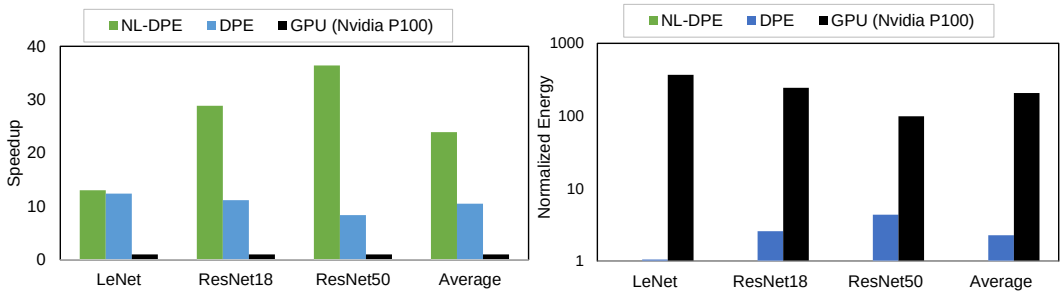
6 Results

Table 1 shows inference accuracy for three neural network models on MNIST and CIFAR-10 datasets, all utilizing DPE for VMM and ACAM for ADCs and activation. We first consider an ideal DPE and ideal activation functions, in which only inputs and outputs are quantized. Then we introduce noise in the ReRAM conductance according to the model previously presented and programming with analog slicing, which results in an accuracy drop. We then implement Noise Aware Finetuning (NAF) as described in Section 5 We then train decision trees to approximate the activation functions and ADCs, we introduce noise for the ACAMs and perform NAF for the ACAMs as well. NAF helps recover the accuracy both for DPE and ACAMs. Note that we did not exhaustively explore all training hyperparameters, the accuracy was largely recovered, suggesting that further optimization of hyperparameters could potentially close this gap. We envision more sophisticated training algorithms for the NL-DPE employing differentiable ACAMs [Pedretti et al. (2022)].

Table 1: Inference accuracy of the proposed NL-DPE, also considering the noise in DPE and ACAM.

	LeNet MNIST	**ResNet18** Cifar10	**ResNet50** Cifar10
Baseline (FP32)	99.15	92.59	92.78
Qunatized Activations	99.15	92.43	92.86
+ Noisy DPE	99.117	89.99	89.03
+ NAF for Noisy DPE	99.133	90.24	90.23
+ ACAM	99.067	90.09	89.69
+ Noisy ACAM	98.483	89.13	87.53
+ NAF for noisy ACAM	98.917	84.88	86.34

NL-DPE completely removes the ADC and digital activation unit in conventional DPE accelerators. We compare the power and area between our proposed NL-DPE computing unit and conventional DPE-based computing unit (including ADC and activation units needed by DPE) in Table 2. We take the ADC data from [Shafiee et al. (2016)] and scaled down to 28nm to match the technology node in NL-DPE. For digital activation unit, we take the data from FlexSFU [Reggiani et al. (2023)], a piecewise linear (PWL) based implementation to approximate arbitrary activation in digital domain. While



(a) Speedup over GPU.

(b) Normalized energy.

Figure 6: Speedup and normalized energy of running the benchmarks on different accelerators.

ACAM's area is larger than that of ADC, it is significantly smaller than the FlexSFU. Considering the whole computing unit, NL-DPE's area and power are both lower than that of DPE based computing.

Table 2: Power and area of ACAM, ADC and digital activation unit.

	Params	**Spec**	**Power** (mW)	**Area** (mm^2)
NL-DPE computing unit				
DPE	array size number	256×256 1	0.82	1.73
ACAM	array size number	16×1 8	0.18	0.009
Total			1	1.74
DPE computing unit				
DPE	array size number	256×256 1	0.82	1.73
ADC	resolution frequency	8bits 1.2GSps	1.71	0.001
FlexSFU	bit width	8bits	3.7	0.015
Total			6.23	1.75

The speedup and energy of running the whole model on three different accelerators are shown in Fig. 6. DPE is a conventional in-memory computing accelerator with ADC and digital activation units [Shafiee et al. (2016)], while NL-DPE replaces ADC and digital activation units with the proposed ACAM approach. We also test on NVidia P100 as the GPU baseline. Larger models have higher speedup and energy saving due to more activation can be paralleled using ACAM. On average, NL-DPE achieves 23× speedup and over 200× energy saving over GPU. Compared with DPE, NL-DPE achieves 2× speedup and 2.5× energy saving on average.

7 Conclusion

We presented the NL-DPE, a novel ADC-less analog in-memory computing primitive. NL-DPE builds on previous works on crossbar arrays and ACAM, by connecting them in the analog domain, with the former accelerating dot products and the latter activation functions, effectively implementing $f(xW)$ in the analog domain. Activations are approximated with decision trees which the ACAM can efficiently accelerate. We propose a Noise Aware Finetuning (NAF) routing to increase accuracy in the presence of ReRAM analog noise and we benchmark the proposed accelerator against the conventional DPE and GPU, reaching significant improvements. We envision NL-DPE as a new building block for in-memory computing, opening up new possibilities for model design fully exploiting the programmable analog non-linearity operation.

References

Ankit Aayush, Hajj Izzat El, Chalamalasetti Sai Rahul, Ndu Geoffrey, Foltin Martin, Williams

R Stanley, Faraboschi Paolo, Hwu Wen-mei W, Strachan John Paul, Roy Kaushik, others . PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference // Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems. 2019. 715–731.

Ielmini Daniele. Resistive switching memories based on metal oxides: mechanisms, reliability and scaling // Semiconductor Science and Technology. 2016. 31, 6. 063002.

Li Can, Graves Catherine E, Sheng Xia, Miller Darrin, Foltin Martin, Pedretti Giacomo, Strachan John Paul. Analog content-addressable memories with memristors // Nature communications. 2020. 11, 1. 1638.

Mao Ruibin, Wen Bo, Jiang Mingrui, Chen Jiezhi, Li Can. Experimentally-validated crossbar model for defect-aware training of neural networks // IEEE Transactions on Circuits and Systems II: Express Briefs. 2022. 69, 5. 2468–2472.

Momeni Ali, Rahmani Babak, Scellier Benjamin, Wright Logan G, McMahon Peter L, Wanjura Clara C, Li Yuhang, Skalli Anas, Berloff Natalia G, Onodera Tatsuhiro, others . Training of Physical Neural Networks // arXiv preprint arXiv:2406.03372. 2024.

Pedretti Giacomo, Ambrosi Elia, Ielmini Daniele. Conductance variations and their impact on the precision of in-memory computing with resistive switching memory (RRAM) // 2021 IEEE International Reliability Physics Symposium (IRPS). 2021a. 1–8.

Pedretti Giacomo, Graves Catherine E, Serebryakov Sergey, Mao Ruibin, Sheng Xia, Foltin Martin, Li Can, Strachan John Paul. Tree-based machine learning performed in-memory with memristive analog CAM // Nature communications. 2021b. 12, 1. 5806.

Pedretti Giacomo, Graves Catherine E, Van Vaerenbergh Thomas, Serebryakov Sergey, Foltin Martin, Sheng Xia, Mao Ruibin, Li Can, Strachan John Paul. Differentiable content addressable memory with memristors // Advanced electronic materials. 2022. 8, 8. 2101198.

Pedretti Giacomo, Mannocci Piergiulio, Li Can, Sun Zhong, Strachan John Paul, Ielmini Daniele. Redundancy and analog slicing for precise in-memory machine learning—Part I: Programming techniques // IEEE Transactions on Electron Devices. 2021c. 68, 9. 4373–4378.

Pedretti Giacomo, Moon John, Bruel Pedro, Serebryakov Sergey, Roth Ron M, Buonanno Luca, Gajjar Archit, Ziegler Tobias, Xu Cong, Foltin Martin, others . X-TIME: An in-memory engine for accelerating machine learning on tabular data with CAMs // arXiv preprint arXiv:2304.01285. 2023.

Reggiani Enrico, Andri Renzo, Cavigelli Lukas. Flex-sfu: Accelerating dnn activation functions by non-uniform piecewise approximation // 2023 60th ACM/IEEE Design Automation Conference (DAC). 2023. 1–6.

Shafiee Ali, Nag Anirban, Muralimanohar Naveen, Balasubramonian Rajeev, Strachan John Paul, Hu Miao, Williams R Stanley, Srikumar Vivek. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars // ACM SIGARCH Computer Architecture News. 2016. 44, 3. 14–26.

Sheng Xia, Graves Catherine E, Kumar Suhas, Li Xuema, Buchanan Brent, Zheng Le, Lam Sity, Li Can, Strachan John Paul. Low-conductance and multilevel CMOS-integrated nanoscale oxide memristors // Advanced electronic materials. 2019. 5, 9. 1800876.

Song Wenhao, Rao Mingyi, Li Yunning, Li Can, Zhuo Ye, Cai Fuxi, Wu Mingche, Yin Wenbo, Li Zongze, Wei Qiang, others . Programming memristor arrays with arbitrarily high precision for analog computing // Science. 2024. 383, 6685. 903–910.

Zhu Hanqing, Zhu Keren, Gu Jiaqi, Jin Harrison, Chen Ray T, Incorvia Jean Anne, Pan David Z. Fuse and mix: MACAM-enabled analog activation for energy-efficient neural acceleration // Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design. 2022. 1–9.

A Reconstruction of Gray coded output

Conversion from Gray code to binary can be achieved using simple logic gates: each binary bit is obtained by performing an XOR operation with all the higher-order bits in the Gray code representation, except the MSB which is the same between these two formats, i.e.,

$$b_i = \begin{cases} g_i & i = n-1 \\ XOR(g_{n-1}, g_{n-2}, ..., g_{i+1}) & i < n-1 \end{cases}$$

where b_i and g_i are the ith bit in the binary and Gray code format. n is the output bit width. The overhead of these XOR gates are inlcuded in the power and area estimation of ACAM.

B ReRAM Noise Model

Since weight values are represented by ReRAM conductance, the conductance should ideally remain stable and exhibit linear properties. However, real ReRAM devices exhibit various stochastic behaviors, such as inaccurate programming, conductance relaxation, and conductance fluctuation [Pedretti et al. (2021a)]. Inaccurate programming refers to the error between the actual conductance and the target value, determined by the programming tolerance. After programming, the conductance may drift before stabilizing—a phenomenon known as conductance relaxation. Conductance fluctuation is temporary noise that occurs during each read operation. Both relaxation and fluctuation are conductance-dependent. For example, evidence shows a constant standard deviation of fluctuations for high conductance states, with a linear decrease once a strong filament is formed and a low resistance state is reached [Ielmini (2016)].

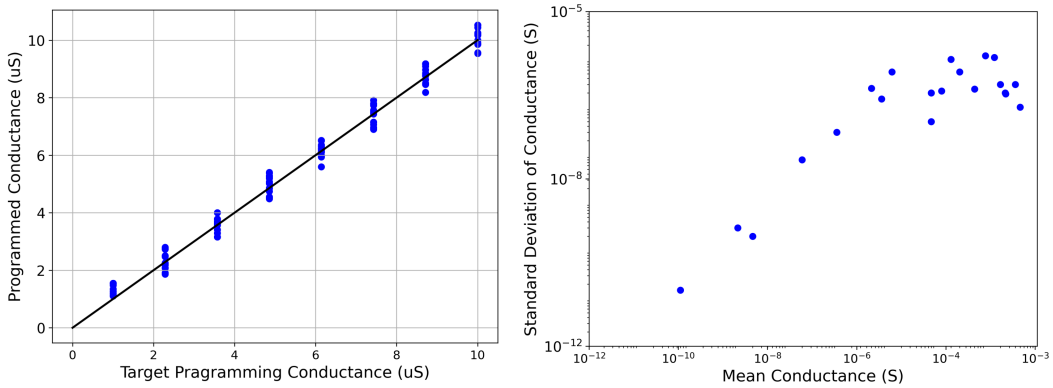


Figure 7: **(a)** Pragramming 8 levels at 10μS and below, with equal conductance spacing of 1.29μS and a fixed tolerance of 0.55μS. **(b)** Standard deviation of read conductance as a function of mean conductance.

We use the same mathematical model from [Mao et al. (2022)] that accounts for all three effects—inaccurate programming, conductance relaxation, and conductance fluctuation—and fit the model parameters using data from TaOx based ReRAM [Sheng et al. (2019)]. Fig. 7(a) shows the actual conductance values programmed at 10μS or below, with 8 levels spaced evenly at 1.29μS and a fixed tolerance of 0.55μS. Fig. 7(b) illustrates the standard deviation (σ) of the read conductance as a function of the mean programmed conductance, demonstrating that while σ increases with the mean, it remains within 1μS across the entire range.

The conductance read out from the device is modeled as:

$$\mathbf{G_{read}} = \mathbf{G_{target}} + \mathbf{G_{relax}} + \mathbf{G_{fluc}} \tag{2}$$

$\mathbf{G_{target}}$ is target conductance converted from weight value w, i.e., $\mathbf{G_{target}} = w \cdot \mathbf{g_{ration}}$, we use $\mathbf{g_{ration}} = 75$. $\mathbf{G_{relax}}$ and $\mathbf{G_{fluc}}$ are the noise caused by relaxation and fluctuation, respectively. After fitting the data, $\mathbf{G_{relax}}$ and $\mathbf{G_{fluc}}$ are computed as:

$$\mathbf{G_{relax}} = N(0.064, 0.31^2) \tag{3}$$

$$\mathbf{G_{fluc}} = exp(log(\mathbf{G_{relax}} + \mathbf{G_{target}}) \cdot 0.75 - 2.93 + N(0, 0.98^2)) \cdot N(0, 1) \tag{4}$$

$N(\mu, \sigma^2)$ denotes the gaussian with mean μ and standard devivation σ.