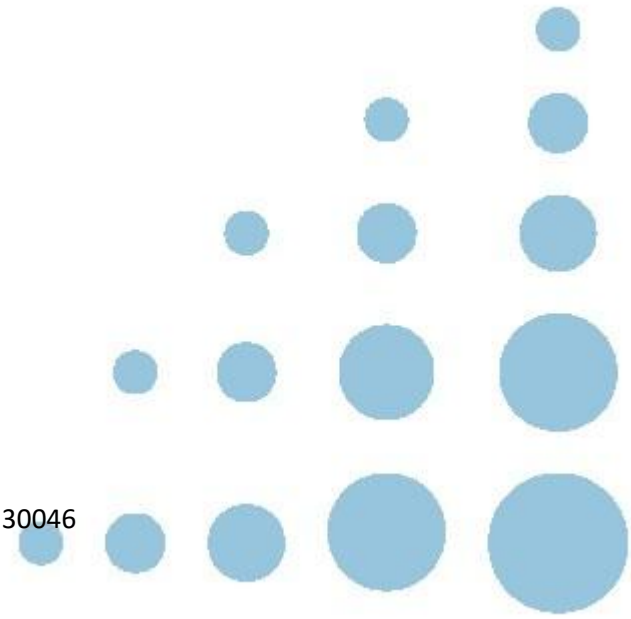
MULTISPIN.AI
AI Co-Processor Research Project

# D4.2
# Standard ML tasks



# Deliverable Details

**Lead beneficiary:** **UCLouvain**

**Work package:** WP4

**Dissemination level:** *PU (Public): Fully open (automatically posted online) / SEN (Sensitive): Limited under the conditions of the Grant Agreement*

**Nature:** *R: Document, report (excluding the periodic and final reports)*

**Due date:** **31/01/2026 (M24)**

**Submission date:** **28/01/2026 (M24)**

**Resubmission date:**

**Authors:** **Pr. Flavio Abreu Araujo, Anatole Moureaux and Anthony Lopes Temporao, UCLouvain**

**Contributors:**

**Reviewers:** **BIU**

**Version History:**

| Date | Version No. | Author | Notes | Pages (no.) |
|---|---|---|---|---|
| 28/01/2026 | 1.0 | Flavio ABREU ARAUJO | Final version for submission | |

Funded by the European Union

**Disclaimer:** *Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union, the European Innovation Council or the SMEs Executive Agency (EISMEA) Neither the European Union nor the granting authority can be held responsible for them.*

# Project Details

Acronym: **MultiSpin.AI**

Title: **n-ary spintronics-based edge computing co-processor for artificial intelligence**

Coordinator: Bar Ilan University (Israel)

Reference: 101130046

Type: HORIZON EIC Grants

Program: HORIZON EUROPE

Topic-ID: HORIZON-EIC-2023-PATHFINDEROPEN-01-01

Start: 01.02.2024 – 31.01.2027

Duration: 36 months

Consortium:

| Id | Participant Name | Short name | Country |
|---|---|---|---|
| 1 | BAR ILAN UNIVERSITY | BIU | ISRAEL |
| 2 | INESC MICROSISTEMAS E NANOTECNOLOGIAS - INSTITUTO DE ENGENHARIA DE SISTEMAS E COMPUTADORES PARA OS MICROSISTEMAS E AS NANOTECNOLOGIAS | INESC MN | PORTUGAL |
| 3 | UNIVERSITE CATHOLIQUE DE LOUVAIN | UCLouvain | BELGIUM |
| 4 | SPINEDGE LTD | SPINEEDGE | ISRAEL |
| 5 | INTERACTIVE FULLY ELECTRICAL VEHICLES SRL | IFEVS | ITALY |
| ~~6~~ | ~~VRIJE UNIVERSITEIT BRUSSEL~~ | ~~VUB~~ | ~~BELGIUM~~ |
| 7 | AMIRES SRO | AMI | CZECHIA |

# Table of Contents

# Executive Summary

This report explores the implementation and evaluation of a n-ary AI inference framework using multistate magnetic tunnel junctions ($M^2TJs$) as core components in analog in-memory computing (IMC) architectures for neural network inference. As traditional von Neumann architectures face limitations due to the memory-computation separation, IMC offers a promising alternative by performing matrix-vector multiplications (MVMs) directly within memory arrays, thereby reducing data transfer overhead and enhancing energy efficiency.

The report focuses on the use of $M^2TJ$-based crossbar arrays, provides a theoretical formulation of the signal retrieval process in such arrays, and examine the analog MAC operation in detail. We implement a quantization method that maps full-precision weights to a limited set of equidistant resistance states, optimized through a sequential least squares search method. Evaluation on the MNIST dataset shows that the inference of the network in a crossbar array of 4-states $M^2TJs$ and without dimensionality reduction achieves 94.48% accuracy. By applying PCA to reduce the input dimensionality, the number of required operations is reduced by 87.44% while accuracy increases to 95.74%, approaching the software baseline at 98.04%.

The robustness of our framework was evaluated on a simulated $M^2TJ$ crossbar, considering three main sources of error: quantization, systematic nonidealities, and cell-specific variations. Quantization error, resulting from the limited number of states per $M^2TJ$, decreases as the number of states increases and is smaller for low-dimensional inputs, indicating that network size and preprocessing influence precision requirements. Systematic nonidealities, such as non-equidistant resistance levels, introduce a uniform bias across the array, while cell-specific variations, including device-to-device differences and noise, contribute deviations that average out across the crossbar. By analyzing these error sources together, we can predict their combined effect on inference accuracy. This analysis also allows to obtain the preliminary formulation of an optimal number of resistance states per $M^2TJ$ for a given noise environment, providing guidance for crossbar design to balance precision, robustness, and hardware constraints.

Our results demonstrate that our framework for n-ary AI inference based on $M^2TJ$ crossbars can support efficient neural network inference with competitive accuracy, provided that quantization schemes, the choice of an optimal number of states per cell, and physical non-idealities are carefully accounted for during design. These findings will be communicated in an article soon to be submitted to either *Applied Physics Letters: Machine Learning* or *Nature Communications Engineering*.

# 1 Introduction

This report presents a framework for neural network inference using a crossbar-based analog in-memory computing (IMC) architecture composed of multistate magnetic tunnel junctions ($M^2TJs$). The motivation for IMC is introduced in the context of current machine learning hardware limitations, followed by a description of matrix–vector multiplication simulated through $M^2TJ$ crossbar arrays. Inference is performed using analog MAC operations and a signal retrieval method to obtain the computed outputs. We detail the quantization strategy used to map trained weights onto discrete resistance levels and validate the approach through XOR and MNIST inference tasks. For MNIST, principal component analysis (PCA) is applied to reduce input dimensionality and limit quantization-induced noise. A detailed error assessment is conducted to analyze quantization error, systematic variations, and cell-specific noise, as well as their combined impact on inference accuracy. This analysis also enables the preliminary determination of an optimal number of resistance states per $M^2TJ$ cell as a function of the noise level. The results illustrate the trade-offs between precision, robustness, and hardware complexity in multistate IMC architectures.

# 2 Background

## 2.1 Machine learning and resources management at the end of 2025

Artificial neural networks (ANN) are mathematical functions with many of parameters called weights. The value of these weights is determined so that the network fills a specific purpose (such as classifying data, predicting quantities, or generating content) with a minimal error. This is done using well-known techniques called error backpropagation and gradient descent, during a phase referred to as the *training* of the network. Once the network is trained, the value of the weights stays fixed and the network can be used to process new pieces of data during a phase known as the *inference*, which is the actual use of the trained ANN. Inferring a standard ANN mainly relies on two simple mathematical operations repeated many times. The first one is the application of a nonlinear activation function (reLU, sigmoid etc) emulating the artificial neurons' output. The other one is a weighted sum that allows to propagate the signal from one layer of neurons to the next one. This weighted sum scales the output signal of the neurons in a layer and sums them to form the input signal for the neurons of the next layer. Despite its apparent simplicity, the so-called multiply-and-accumulate (MAC) operation, also often more pragmatically referred to as matrix-vector multiplication (MVM), is the cornerstone of any ANN.

However, modern computers, whose architecture separates the processing unit (CPU) from the memory (RAM), struggle to perform so many (although simple) operations efficiently as it requires moving data back and forth repeatedly between both units. This phenomenon leads to an increased processing time and energy consumption, and is further worsened by the von Neumann bottleneck, or memory wall, which depicts the increasing gap between the time required by the CPU to process data, and the much longer time required to retrieve said data from the RAM. Therefore, the CPU spends an increasing amount of time waiting for the data to be read from the memory unit [1]. This sparked the development of a new generation of components for running intelligent systems, namely graphic processing units (GPU) [2] and tensor processing units (TPU) [3]. However, since the end of 2025, the ever-increasing use of AI worldwide has put the RAM manufacturing sector under pressure, leading to the explosion of the prices for all kinds of smart electronic devices, from laptop and smartphones to vacuum cleaners and connected fridges [4]. The energy consumed by AI data centers has not stopped increasing either, even triggering the planning of new dedicated nuclear reactors in some cases [5]. Finally, the amount of water consumed for the cooling of data centers has also recently raised concerns regarding the sustainability of AI [6].

## 2.2 In-memory computing

These observations have justified the need for the research and development of new memory technologies and frameworks optimized for AI applications. Among them, in-memory computing (IMC) proposes to perform MVMs directly in the RAM, solving the von Neumann bottleneck and the memory wall issues for these operations. More specifically, the network's weights learned during the training phase are stored in a crossbar array of RAM cells under the form of resistance (resp. conductance) levels (Fig. 1). The input vectors, injected under the form of current intensities (resp. voltages), are analogically multiplied by the resistance (resp. conductance) value in each cell to generate a voltage (resp. current). The generated voltages (resp. currents) are then summed along the transverse direction accordingly to Kirchhoff's law to perform the accumulation. The output voltages (resp. currents) hence contain the result of the MVM [7]. While this concept is not new, most of the implementations proposed so far must accommodate with significant constraints that hinder their efficiency. This is due to the complexity of implementing a MVM analogically with minimal error. To our knowledge, most of the proposed solutions rely on devices with two resistance states, hence constraining the weights to binary encoding and limiting the computational power of the system [8-13]. Some works circumvent this issue by combining several binary devices in a single crossbar cell to emulate a larger number of bits, but at the expense of the size and the complexity of the system [9, 14-15]. While other works report a multibit encoding of the weights in the crossbar array, some kind of quantization is required in the encoding of the input and output signal, once again limiting the computational performance of the solution [16]. Hence, a straightforward solution allowing the analogue multiplication of continuous input signals with multibit weights is still missing.

In the scope of this project, we developed a method allowing the computation of analogue MVMs in a crossbar array of multistate magnetic tunnel junctions ($M^2TJ$). We aim at providing a simple and generalizable framework that enables the use of standalone multistate devices with minimal assumptions, as well as providing insights on the expected performance of said systems and an assessment of the different sources of error.

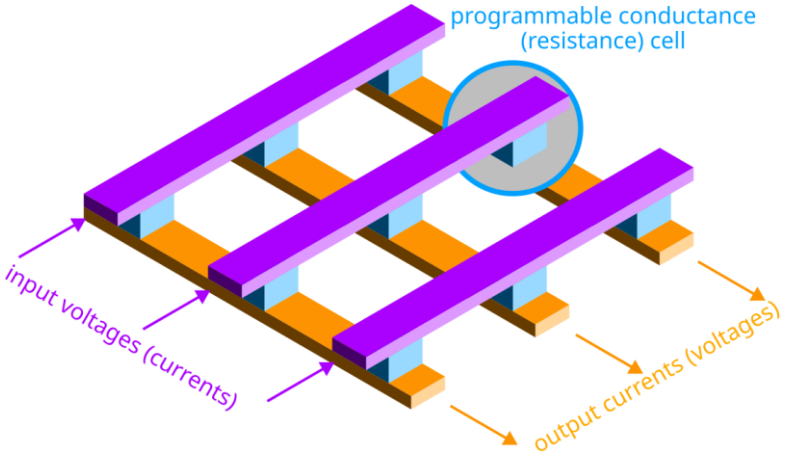



**Figure 1: A crossbar array of cells with programmable conductance (resistance) states allows the computation of a MVM in the memory.**

## 2.3 Simulating MVMs using a crossbar array of $M^2TJs$

In the previous period of the project, we presented a method for encoding weights into $N$ resistance values, and input signals into current intensities. Note that this method can also be used in the reverse configuration, *i.e.* with weights encoded into conductance values and with input signals encoded into voltages. We will however stick to the resistance configuration for the sake of readability.
Our goal is to perform the following MVM

$$y = Wx$$

To do so, we must first quantize the weights matrix $W$ into $N$ equidistant values $A_i$ with minimal error. This quantization step is mandatory, due to the discrete number of resistance levels in each M²TJ. As a base case, we considered $N = 4$ but the approach is generalizable to any $N$ value. To do so, we minimize the squared distance between the $N$ values $A_i$ and the weights $w$ from matrix $W$ that are the closest to $A_i$, a problem similar to the K-means optimization problem, with an additional constraint of equidistance between the centroids. The quantized weight matrix $A$ is then described as follows:

$$\text{argmin}_A \sum_{i=1}^{N} \sum_{w \in A_i} |w - A_i|^2 \text{ with } A_i - A_{i-1} = d$$

The MVM to be computed using the crossbar array is thus

$$\tilde{y} = Ax$$

Then, we assume a linear scaling of the input vector $x$ into a vector of current intensities values $I$ with coefficients $a_I$ and $b_I$ defined accordingly to the requirements of the system:

$$I = a_I\, x + b_I$$

A similar scaling can be used to map the elements of the quantized weights matrix A to the $N$ resistance levels of the M²TJs, hence forming the resistance matrix R:

$$R_{mn} = a_R\, A_{mn} + b_R$$

The voltage generated in the cell at position $(m, n)$ in the crossbar array is thus equal to

$$V_{mn} = R_{mn}\, I_n$$

and the voltage accumulated to form one element of the output vector is equal to

$$V_m = \sum_n V_{mn} = \sum_n R_{mn}\, I_n$$

which after substitution yields

$$V_m = a_R\, a_I \sum_n A_{mn}\, x_n + b_R\, a_I \sum_n x_n + a_R\, b_I \sum_n A_{mn} + n\, b_R\, b_I$$

We can also write

$$\tilde{y}_m = \sum_n A_{mn}\, x_m$$

Hence,

$$y_m = \frac{V_m - b_R\, a_I \sum_n x_n - a_R\, b_I \sum_n A_{mn} - n\, b_R\, b_I}{a_R\, a_I}$$

which under vectorial form simplifies to:

$$\tilde{\boldsymbol{y}} = \frac{\boldsymbol{V} - b_R(\boldsymbol{I}^T \cdot \mathbf{1}) - a_R b_I(\boldsymbol{A} \cdot \mathbf{1})}{a_R a_I}$$

As a result, the approached results of the MVM ỹ can be retrieved by scaling the measured signal $V$ by removing the two constant offsets $\left(b_R\,(I^{\mathsf{T}} \cdot 1)\right)$ and $\left(a_R b_I (A \cdot 1)\right)$ and then dividing by the constant factor $(a_R\, a_I)$. The term $(I^{\mathsf{T}} \cdot 1)$ is simply a scalar value representing the sum of the input signal, which can be computed easily on chip or stored in the memory of the measurement equipment. The term $(A \cdot 1)$ is a vector containing the row-wise sum of $A$, which must be computed only once and then can be stored in the memory of the measurement equipment as well.

In most of the cases, input data is scaled in [0, 1] before being fed to an ANN for standard machine learning tasks. Hence, by setting $b_I = 0$, the input signal is simply scaled by the parameter $a_I$ which represents the maximum

current intensity that one can inject in the M²TJs. This simplified approach with yields a simpler version of the scaling of the measured signal used to retrieve the result of the MVM:

$$\tilde{\boldsymbol{y}} = \frac{\boldsymbol{V} - b_R(\boldsymbol{I}^T \cdot \mathbf{1})}{a_R a_I}$$

In the previous reporting period, we used this method successfully to perform MVMs in a simulated crossbar array of (4x4) M²TJs with 4 levels of resistance, with continuous input signals. It has also been showed that it could be scaled to larger arrays and numbers of states easily.

# 3 Performing standard ML tasks with the crossbar array

Since the development of the method presented above, we progressed concurrently on computing MAC operations and performing standard machine learning (ML) tasks using a simulation of the M²TJs crossbar array. In this section we present the results of the second deliverable of work package 4, *i.e*. the computation of standard machine learning tasks using a (simulated) crossbar array of M²TJs. We first tackle a very simple ML task to present in detail the procedure used to perform the inference with the crossbar array. We will then focus on presenting the results of a more complex ML task in a following section.

## 3.1 The XOR approximation problem

We first tackle a simple yet important task in the field of ML. The XOR approximation problem is one of the smallest problems requiring a multilayer perceptron (MLP) to be solved. It consists in training a small neural network to approximate the XOR function, *i.e.* to generate the same output as the XOR function for any input. Performing the inference of the trained network with the crossbar array and the method presented above will ensure that the key characteristics of a simple neural network are conserved in the hardware.

### 3.1.1 Approximating the XOR function

The exclusive OR function (XOR) is a binary function whose truth table is presented in Fig. 2, left. To successfully approximate XOR, a neural network needs to be able to learn a "boundary" separating the input space into two regions, each of them yielding a specific output (0 or 1). In other words, the network must emulate a border separating (0, 0) and (1, 1) from (0, 1) and (1, 0) (Fig. 2, right). This border cannot be a single line, but some sort of nonlinear curve. That's why the XOR approximation problem allows to assess the ability of the network to emulate a nonlinear function, and by extension its universal approximation ability. While the only valid inputs are binary ((0, 0), (0, 1), (1, 0), (1, 1)), one can also feed inputs continuously arranged between 0 and 1 into an ANN trained to approximate XOR. This helps visualize the boundary learned by the model during the training phase. We evaluate the performance of the crossbar array inference by comparing the output array for continuous entries in $[0,\ 1] \times [0,\ 1]$ yielded by the software ANN and the hardware version.

| XOR | | |
|---|---|---|
| $x_1$ | $x_2$ | y |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

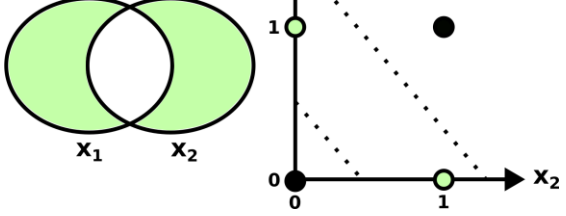



**Figure 2: Left: Truth table of the XOR function. Center: XOR Venn diagram. Right: Input space and output values of the XOR function.**

We trained an ANN with 2 input neurons, 2 hidden neurons, and 1 neuron, all activated with the sigmoid function. The training was carried for 2000 epochs using the binary cross entropy loss function, and the Adam optimizer with a learning rate of 0.1. The trained network and the output obtained for inputs in [0, 1] are presented in Fig. 3 and Fig. 4.

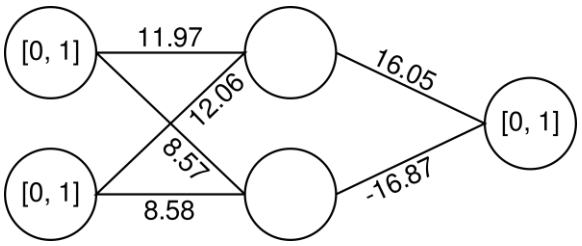



**Figure 3: Network trained for the XOR task. The biases are not represented.**

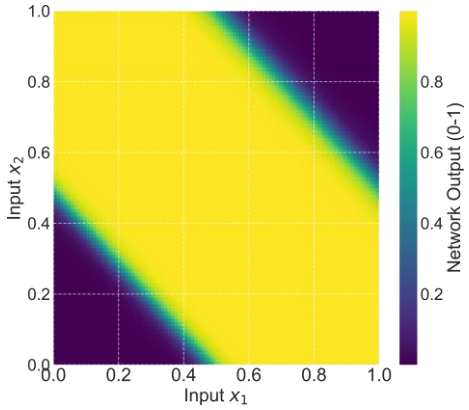



**Figure 4: Output map of the neural network trained for solving the XOR task. Notice the similarities with Fig. 2, right.**

### 3.1.2 Inference using the crossbar array

In this section we present how we used a simulated crossbar array of (4 x 4) M²TJs with 4 resistance levels to generate the same output as in Fig. 4.

We first set $a_I = 0.0005$ to scale the input between 0 and 0.5 mA. For the record, the value of the switching current reported in Ref. [17] is 10 mA, but the reading current can be much lower. The scaling is represented in Fig. 5.

$$I = 0.5\,x \text{ (mA)}$$

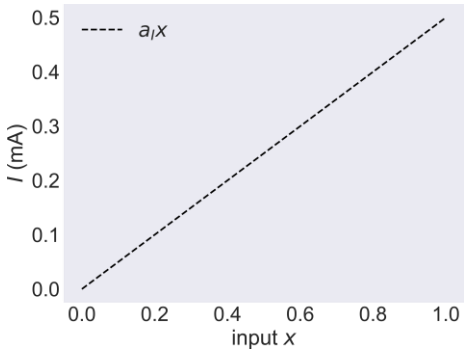



**Figure 5: Scaling of the input data.**

Then, we must quantize the weights matrices learned in the previous section into 4 values. The weights values in the first layer are the following:

$$\{8.57,\ 8.58,\ 11.97,\ 12.06\}$$

The 4 equidistant values obtained with our quantization method are the following:

$$\{8.58,\ 9.72,\ 10.87,\ 12.01\}$$

Hence, the weights matrix of the first layer $W_1$ is quantized accordingly to form matrix $A_1$:

$$\begin{bmatrix} 11.97 & 12.06 \\ 8.57 & 8.58 \end{bmatrix} \rightarrow \begin{bmatrix} 12.01 & 12.01 \\ 8.58 & 8.58 \end{bmatrix}$$

Note that $A_1$only contains two distinct values because some of the values in $W_1$ are originally close (11.97 and 12.06, and 8.57 and 8.58). The quantization values of $A_1$ are then mapped to 2 of the 4 resistance values reported in Ref. [17] with the scaling coefficients $(a_R, b_R) = (150.59, 8110.40)$ as represented in Fig. 6. The resistance values of two of the cells are thus programmed to be 9402 Ω, and two others are programmed so that their resistance is 9919 Ω. The output signal measured from the simulated crossbar array is then converted to obtain $\tilde{y}$ and the biases learned during the training are added. The sigmoid function is applied to the result to form the input signal $x$.

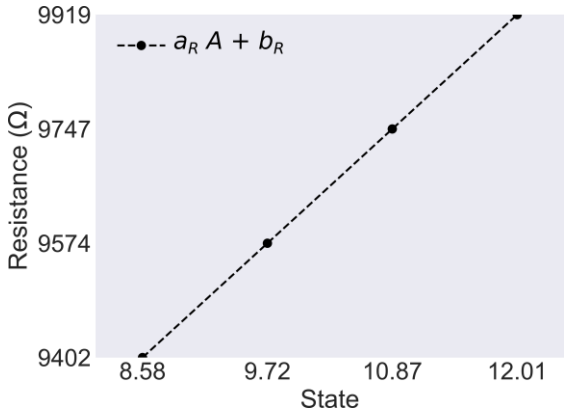



**Figure 6: Scaling of the weights into resistance states.**

A second MVM must then be performed to transmit the signal to the neuron of the last layer of the network. We choose the scaling coefficients $a_I$ and $b_I$ so that the input signal is still in the $[0,\ 0.5\ \text{mA}]$ range. As the weights matrix of the second layer $W_2$ contains only two values ($-16.87$ and $16.05$ ), it does not need quantization, and the two values are mapped to the lowest and highest resistances values (Fig. 7). The simulated output voltage signal is once again scaled to retrieve $\tilde{y}$, and the learned biases are added before applying the last sigmoid function.

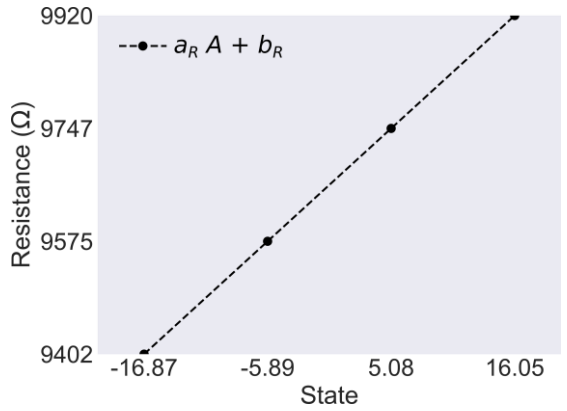



**Figure 7: Scaling of the weights of the second layer into resistance states**

### 3.1.3 Results

The output map obtained using the crossbar array inference is displayed in Fig. 8. It can be directly seen that the appearance of the map is very similar to the one obtained with the classical software approached. The relative difference between the two maps presented in Fig. 9 shows that the error in the crossbar array output (here, the software output is considered as the ground truth) is only localized on the decision borders. In other words, the network inferred with the crossbar array agrees with the software version for the data points that matter ([0, 0], [0, 1], [1, 0] and [1, 1]), and disagrees the most for the data points that are the less relevant. This indicates that the crossbar array inference conserves the principal characteristics of the model learned by the network during the training phase. However, the only source of error considered so far is the quantization process, so additional sources will be assessed in a following section.

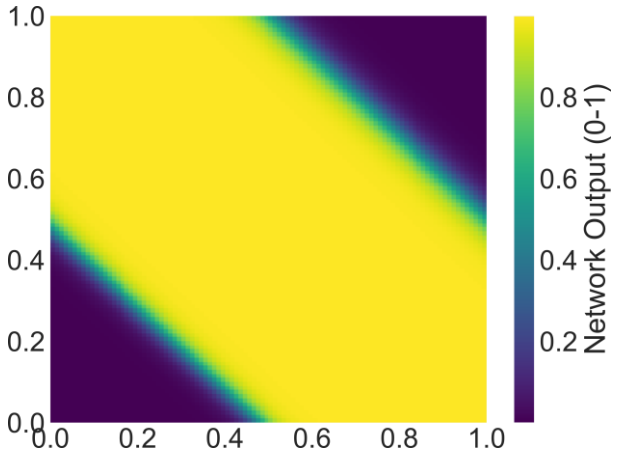



**Figure 8: Output map of the network trained for solving the XOR task and inferred with the simulated crossbar array.**

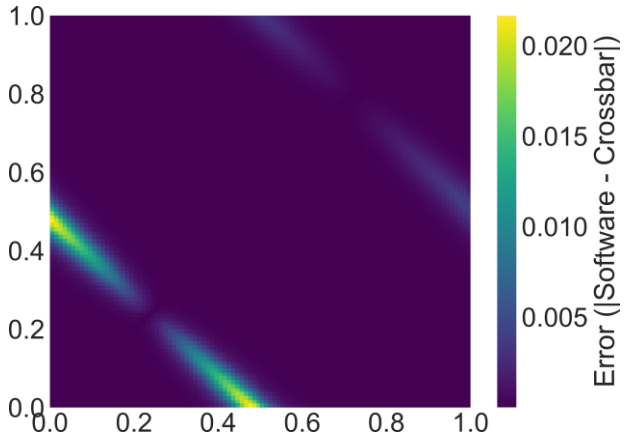



**Figure 9: Absolute difference between the software and the crossbar outputs.**

### 3.1.4 Remarks

#### 3.1.4.1 Scaling values

The coefficients used for scaling the input $x$ into current intensity values ($a_I$ and $b_I$) were chosen as a function of the minimum and maximum values of $x$ to ensure that the scaled signal stayed in the range $[0,\ 0.5\ \text{mA}]$. However, these values are arbitrarily chosen and can be tuned to better fit the actual hardware and the experimental constraints as long as the output signal values stay resolved to be measured accurately.

#### 3.1.4.2 Crossbar array multiplexing

While a $(4 \times 4)$ crossbar array was considered, the two MVMs performed here were involving matrices of size $(2 \times 2)$ and $(1 \times 2)$. The crossbar array can be used for emulating smaller matrices by voluntarily avoiding the use of all the M²TJs. In the case of the $(2 \times 2)$ matrix, two input channels would get zero input signal, and two of the output channels would be discarded for example. Larger matrices can also be emulated using a crossbar array of limited size by leveraging the multiplexing technique presented in Ref. [9]. It consists in using the array several times successively with different sets of weights and recombining the outputs to reconstruct the output of a virtually larger array.

## 3.2 MNIST classification

The XOR approximation problem confirms that the basic properties of a trained network are conserved when the inference is performed in an idealized crossbar array of M²TJs. However, a more complex and concrete ML task is required to get a better insight of the actual performance of the hardware as an AI co-processor. In this regard, we chose to tackle the MNIST classification task, one of the most famous ML tasks in the ML community [18]. This will allow us to assess in more detail the performance of our hardware inference implementation, its limitations, robustness to error, and potential paths for further optimization.

### 3.2.1 The MNIST dataset

The MNIST dataset is composed of 70000 images of handwritten digits from 0 to 9. Usually, 60000 images are used to train the model, and 10000 are used to test its performance. Each image is a $(28 \times 28)$ grid of pixels (Fig. 10) whose intensity is represented by a number between 0 and 255 (most often scaled in the [0, 1] range beforehand).


**Figure 10: Three examples of the MNIST dataset and the corresponding labels**

### 3.2.2 Classifying MNIST

We train a neural network with 784 input nodes (one for each pixel), 128 hidden neurons activated by the reLU function, and 10 output neurons activated by the SoftMax function (Fig. 11). The final output is determined by selecting the index of the output node with the largest value (argmax). The network achieves a classification accuracy of **97.56%**, which is in the average of the expected performance for a simple fully connected ANN.

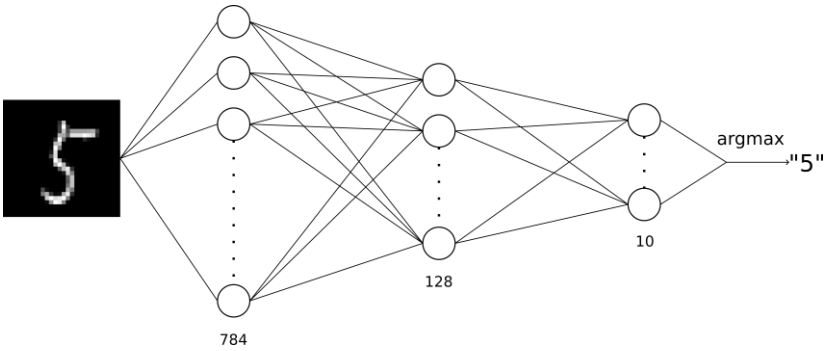



**Figure 11: Network trained for classifying the MNIST dataset.**

### 3.2.3 Inference using the crossbar array

Once again, two MVMs are required to infer the trained network. The first one multiplies the 784-long input vector representing an image with the $(128 \times 784)$ weights matrix of the first layer, and the second MVM multiplies the 128-long hidden vector with the $(10 \times 128)$ weights matrix of the second layer to generate the output vector.

The method used to perform the MVMs is identical to the one used for the XOR approximation task, except that here the $(4 \times 4)$ crossbar array must be used several times to emulate the much larger matrices involved in the computations. This is done using the multiplexing method presented in section 3.1.3.2. However, the emulation of the first MVM alone would require performing 6272 measurements and the second MVM would require 96 measurements, totalizing 6368 operations.

$$\left\lceil \frac{128}{4} \right\rceil \times \left\lceil \frac{784}{4} \right\rceil = 6272$$

$$\left\lceil \frac{10}{4} \right\rceil \times \left\lceil \frac{128}{4} \right\rceil = 96$$

While tremendous, this number of operations is only imposed by the limited size of the crossbar array and hence is expected to decrease as technology develops and improves. However, it can already be improved using *principal components analysis* (PCA). This technique consists in decreasing the dimensionality of the input data while preserving the relevant features (*i.e.,* pixels or combinations of pixels that show the most variance in the data). The principal components are first computed (under the form of vectors of weights) using only the training dataset to avoid introducing a bias in the model. Then, the principal components are extracted from the input data before being fed in the network. This technique allows to reduce the input dimension down to 87 while keeping 90% of the variance (orange curve in Fig. 12), leading to a total number of only 800 operations (87.44% less than originally). It is important to note that the extraction of the principal components from the input data requires performing a

dot product, which is also itself a kind of MVM. But as dot products are involving two vectors (instead of one vector and one matrix), it is probably more worth it to perform it within a digital module close to the crossbar rather than within the crossbar array itself.

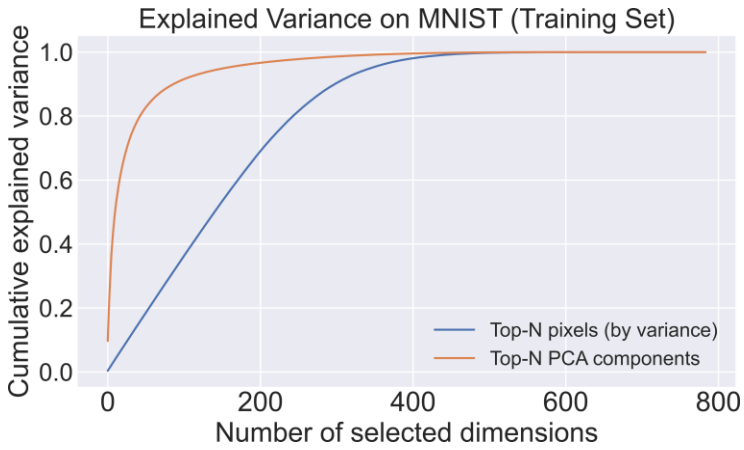



**Figure 12: Explained variance in the MNIST training set as a function of the number of input dimensionality after PCA pre-processing.**

### 3.2.4 Results

Without applying the PCA technique (*i.e.*, using the raw input data) the accuracy reached with a $(4 \times 4)$ crossbar array of M²TJs with 4 states, the accuracy reached by the network is **94.48%**, representing a decrease of **3.18%** from the accuracy reached with the full software inference. This performance degradation is due to the quantization process from 32-bits values to 2-bits values, as no other sources of errors were considered in the simulations so far.

As a side study, the first 87 principal components determined with the PCA technique in the training set were extracted from each sample of the dataset as a pre-processing strategy. The accuracy reached with the full software inference was **98.03%**. This increase in accuracy compared to the case without the PCA pre-processing comes from the fact that PCA effectively removes noisy features from the data by only keeping the relevant (more significant) ones. Moreover, the resulting network is simpler, which helps reaching a better accuracy score. The accuracy reached with the same model inferred with the M²TJs crossbar array is **95.24%**, representing a drop in accuracy of **2.79%** compared to the software solution. While being the same order of magnitude, the accuracy drop between the software and hardware inference processes is 13% lower when using the PCA pre-processing technique. This can be explained by the network architecture, which is simpler in the latter case. This limits the errors introduced by the quantization process as the weights matrix of the input layer is much smaller than without PCA. Simplifying the architecture and reducing the number of weights of the network hence allows to mitigate the impact of quantization errors.

# 4 Error assessment

Here we show a deeper analysis of the different error sources acting on the array, how they combine, and how their impact on the performance of the system scales. We also present a study on determining an optimal number of resistance states in each M²TJ.

## 4.1 Quantization error

The limited number of states in each M²TJ imposes a mandatory loss of precision in the representation of the weights, originally encoded on 32 or 64 bits in the software. The resulting quantization error can be limited by determining appropriate equidistant "quantized" weights values $A_i$, which are optimized by minimizing the distance between each weight $w$ and the closest quantized weight value $A_i$ as presented in section 2.3. The

quantization error is however expected to decrease as $1/N$ where $N$ is the number of states in each M²TJ, as shown in Fig. 13. As already presented in section 3.2.4, the quantization error is mitigated in simpler neural architectures with low-dimensional input data due to its limited accumulation is smaller weights matrices.

## 4.2 Systematic errors

We considered the impact of "systematic" errors to enhance the reliability of the simulations. Systematic errors come from sources that have the same influence on all the devices in the crossbar array. Namely, nonidealities in the fabrication process may lead to a nonideal resistance levels distribution, where said levels are not perfectly equidistant. As a result, all the M²TJs have the same levels distribution where the resistance levels cannot be placed perfectly on a line. This introduces a bias in the measured signal and hence in the computed result of the MAC operation. In the simulations, this nonideal states distribution is represented by adding a random offset drawn from a normal distribution $N(0,\ \sigma_{\mathrm{NL}})$ where $\sigma_{\mathrm{NL}}$ is the root-mean-square error between the nonideal states distribution and the corresponding linearly regressed line (Fig. 14).

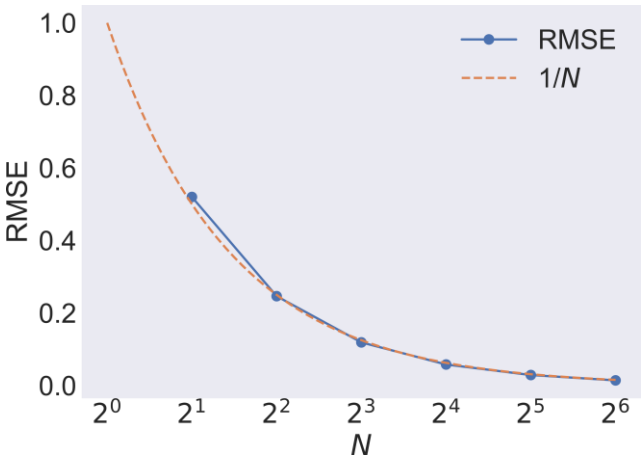



**Figure 13: RMSE in random MVMs performed with a simulated crossbar array of (4 x 4) M²TJs, with respect to the number of states in each M²TJ.**

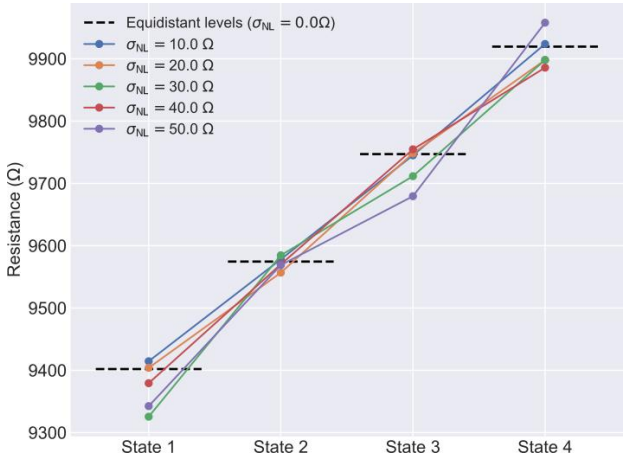



**Figure 14: Ideal equidistant resistance levels (black dashed lines), and nonideal variations obtained by adding random offsets to the levels.**

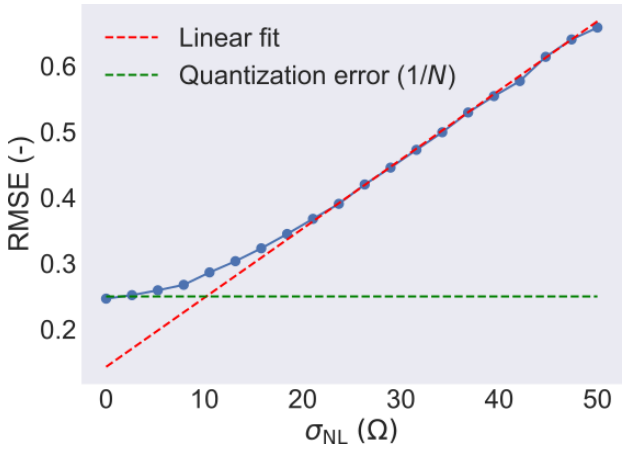



**Figure 15: Average RMSE in the result of MVMs performed with a crossbar array of (4x4) M²TJs with 4 resistance levels for various levels of systematic nonideality $\sigma_{NL}$.**

We simulate random MVMs performed with a crossbar array of $(4 \times 4)$ M²TJs with 4 resistance levels. Figure 15 shows the average RMSE in the result of the MVM computed based on the software ground truth. When $\sigma_{NL} = 0$, the only source of error left is the quantization process and the RMSE reaches a plateau at$1/N = 0.25$. As $\sigma_{NL}$ increases, the influence of the quantization error in the RMSE progressively decreases until the RMSE finally increases linearly with $\sigma_{NL}$.

## 4.3 Cell-specific errors

On top of the quantization error and the nonideal states distribution shared by all the M²TJs in the crossbar array, we consider errors specific to each M²TJ. These cell-specific errors bring together contributions such as cells-to-cells variations and resistance noise. In the simulations, this is also represented by simply adding a random offset drawn from a distribution $N(0, \sigma_{\perp})$ to the resistance levels that were initially linearly distributed. The difference with the previous case (systematic nonidealities) is that in this case the offsets are different from cell to cell, which will lead to averaging over the whole array. As a result, the average RMSE in the MVM result increase less quickly with $\sigma_{\perp}$ than with $\sigma_{NL}$ as shown in Fig. 16, right.

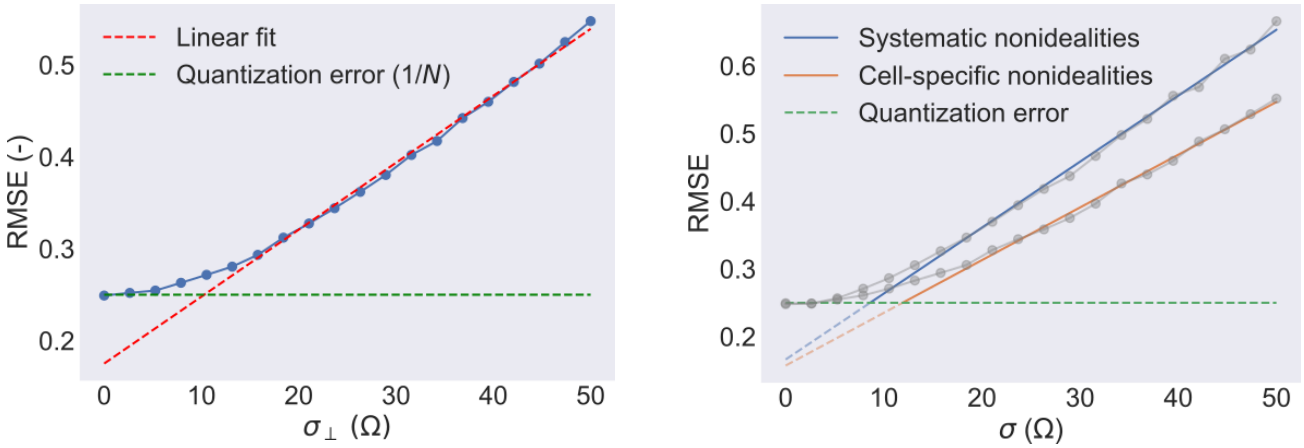



**Figure 16: Left: Average RMSE in the result of MVMs performed with a crossbar array of (4x4) M²TJs with 4 resistance levels for various levels of cell-specific nonideality $\sigma_{\perp}$. Right: Comparison of the RMSE increase with respect to the level of systematic nonideality $\sigma_{NL}$ and the level of cell-specific nonideality $\sigma_{\perp}$.**

## 4.4 Combined error impact

The three sources of error (quantization, systematic nonidealities, cell-specific nonidealities) can be considered together to assess their combined impact on the average RMSE in the result of MVMs (Fig. 17).

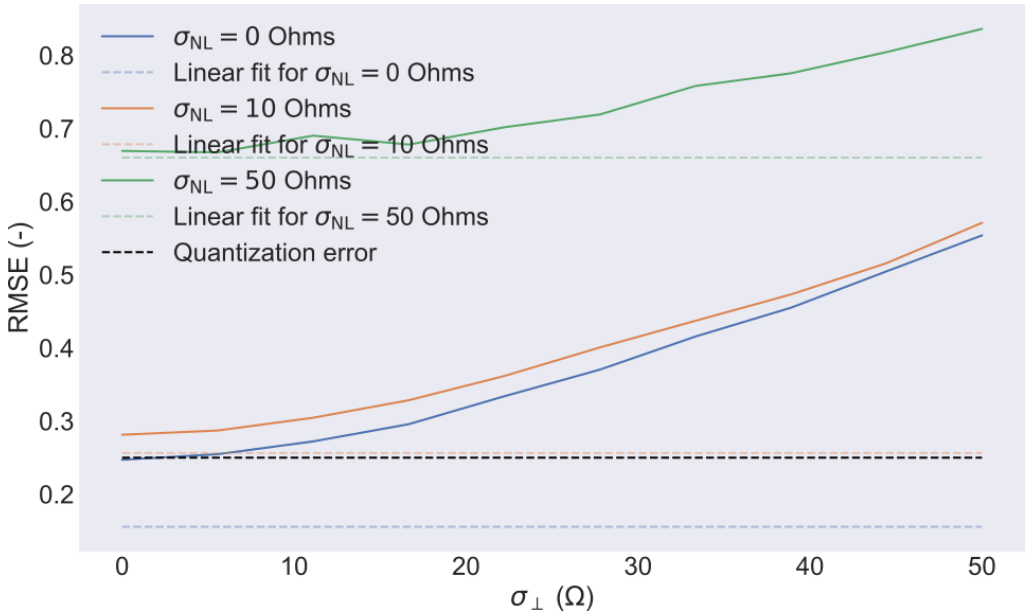



**Figure 17: Evolution of the RMSE in the MVM result as a function of the level of cell-specific nonideality $\sigma_\perp$ for different values of the level of systematic nonideality $\sigma_{NL}$.**

The 3 error contributions can be used to fully explain the combined impact on the RMSE in the MVM results:<

- $\sigma_{NL} = 0\ \Omega$ **(blue curves in Fig. 17):** The only sources of error are the quantization and the cell-specific nonidealities. The curve reaches a plateau at the value of the quantization error $1/N$ (0.25) when $\sigma_\perp = 0\ \Omega$ and ends up increasing linearly as $\sigma_\perp$ increases as in Fig. 16, left.
- $\sigma_{NL} = 10\ \Omega$ **(orange curves in Fig. 17):** The RMSE also reaches a plateau as $\sigma_\perp$ decreases down to $0\Omega$. The value of this plateau is expected to be close to 0.25 if the linear fit in Fig. 15 is considered. However, one can see that the sum of the quantization error and the systematic errors is slightly higher due to the interplay between the two contributions (Fig. 15). Hence, the actual error at low $\sigma_\perp$ values is accordingly closer to 0.3. As $\sigma_\perp$ increases, the linear trend reappears.
- $\sigma_{NL} = 50\ \Omega$ **(green curves in Fig. 17):** The plateau at low $\sigma_\perp$ values is closer to the value of the linear fit presented in Fig. 15 as expected. The linear trend at higher $\sigma_\perp$ values is also observed.

## 4.5 Optimal number of states

Figure 13 may seem to indicate that an ever-increasing number of states per M²TJ would reduce the quantization error. However, the consideration of additional sources of error would induce an increase of the error if $N$ becomes too large. Indeed, squeezing more and more resistance levels between two values $R_{min}$ and $R_{max}$ will lead to a loss of resolution between adjacent levels in the presence of noise leading to deviation from linearly distributed states. Hence, there is an optimal number of states per M²TJ $N_{opt}$, which corresponds to the maximum number of states above which the resistance levels are not resolved anymore, leading to an increase of the error. The optimal number of states $N_{opt}$ should be proportional to the accessible resistance range $R_{max} - R_{min}$ and inversely proportional to the combined level of nonideality in the states distribution $\sqrt{\sigma_{NL}^2 + \sigma_\perp^2}$:

$$N_{opt} \propto k \frac{R_{max} - R_{min}}{\sqrt{\sigma_{NL}^2 + \sigma_\perp^2}}$$

The preliminary results presented in Fig. 18 show that the number of states per M²TJ is effectively decreases as $\sigma_\perp$ increases, for a constant $\sigma_{NL}$ value. More study is needed to define a proper definition of $N_{opt}$ as a function of the different error contributions.

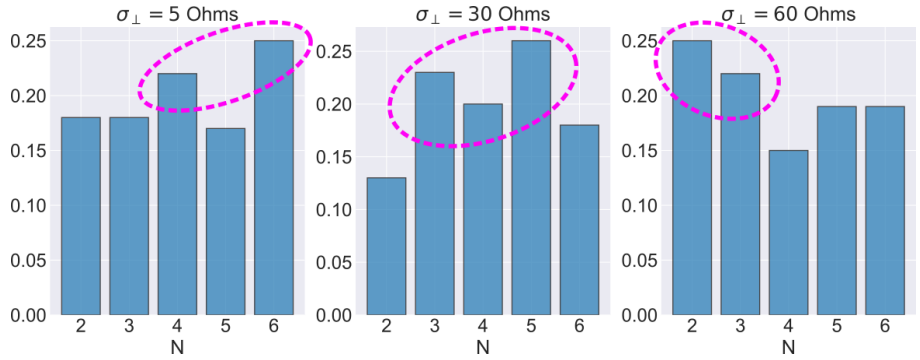



**Figure 18: Histograms representing the relative distribution of the number of states per M²TJ *N* minimizing the RMSE in the result of random MVMs. The value of $\sigma_{NL}$ was fixed to $50\ \Omega$, and $\sigma_\perp$ was fixed to 5 $\Omega$, 30 $\Omega$, and 60 $\Omega$. As $\sigma_\perp$ increases, lower $N$ values yield lower error in the MVM results.**

# 5 Conclusion

To address the ecological concerns raised by the recent surge in the use of AI since 2022, we studied the implementation of a co-processor to perform MVMs in the framework of IMC. The system used here is a crossbar array of (4x4) M²TJs presenting each 4 distinct resistance states. A method was developed to retrieve the result of the MVMs by rescaling the measured output signal appropriately. This method allows to simulate the computation of MVMs using the crossbar array. Standard machine learning tasks are then performed using the inference scheme based on the crossbar array. The XOR approximation task shows that the hardware inference process preserves the characteristic of the model learned during the training phase. Some errors are present due to the quantization of the full-precision weights into only 4 values. The errors are however localized on the decision border of the model, *i.e.* where they are the less problematic. The agreement between the software and the hardware inference processes is almost conclusive on the data points of interest. To assess the performance of the method on a more complex task, we tackled the MNIST classification problem. A naively complex neural network reached 97.56% test accuracy, while the hardware implementation reached 94.48%, marking an accuracy drop of 3.18%. This drop is also due to the quantization process. The limited size of the considered crossbar array requires to use it thousands of times to emulate the whole operations involved in the inference process. By reducing the dimensionality of the input data from 784 to 87 using PCA, the resulting network is much simpler and the number of parameters much smaller. The network achieves 98.03% test accuracy when inferred using software. The hardware inference achieves 95.24% test accuracy, marking a drop of 2.79%. This smaller difference between the software and hardware approaches is due to the mitigation of quantization error in smaller weights matrices and smaller networks. Hence, reducing the dimensionality of the input data is a good strategy for simplifying the experiments while improving the performance. We then assess two other sources of errors: systematic nonidealities that are common to all the devices in the array, and cell-specific nonidealities, which benefit from averaging across the whole array. When these contributions are weak, the error in the result of MVMs is dominated by the quantization error, but it ends up increasing proportionally to the level of nonideality introduced in the resistance states distribution by both sources of error. The combined impact of all three sources of error can be fully explained by the separate behavior of these contributions. Finally, we determine that there exist an optimal number of states per M²TJ that minimizes the error in the result of the MVMs by compromising between the quantization error and the resolution of the resistance states. Preliminary results show an expected decrease of this optimal number of states when the combined level of nonideality in the states distribution due to the two latter sources of error increases.

# 6 Bibliography

[1] Zou, X., Xu, S., Chen, X. *et al*. Breaking the von Neumann bottleneck: architecture-level processing-in-memory technology. *Science China Information Sciences 64, 160404* (2021).

[2] Krizhevsky, A., Sutskever, I., Hinton, G. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25* (2012).

[3] Jouppi, N., Young, C., Patil, N. *et al.* In-Datacenter Performance Analysis of a Tensor Processing Unit. *arXiv:1704.04760* (2017).

[4] Jeronimo, F., Mainelli, T., Ma, B. *et al.* Global Memory Shortage Crisis: Market Analysis and the Potential Impact on the Smartphone and PC Markets in 2026. *IDC* (2025).

[5] de Jongh, Z., Homveld, T., Winters, M. Data center boom empowers modular nuclear energy opportunities. *A&O Shearman* (2025).

[6] Montero, M. From Cloud to Cup: How Much Water Does Your ChatGPT Drink? *IE Insights* (2025).

[7] Xia, Q., Yang, J.J. Memristive crossbar arrays for brain-inspired computing. *Nature Materials 18, 309–323* (2019).

[8] Zhang, Y., Cui, M., Shen, L. *et al.* Memristive Quantized Neural Networks: A Novel Approach to Accelerate Deep Learning On-Chip. *IEEE Transactions on Cybernetics 51, 4, 1875-1887* (2021).

[9] Jung, S., Lee, H., Myung, S. *et al*. A crossbar array of magnetoresistive memory devices for in-memory computing. *Nature 601, 211–216* (2022).

[10] Li, S., Niu, D., Malladi, K. *et al.* DRISA: a DRAM-based Reconfigurable In-Situ Accelerator. *MICRO-50 '17* (2017)

[11] Seshadri, V., Lee, D., Mullins, T. *et al*. Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology. *MICRO-50 '17* (2017).

[12] Doevenspeck, J., Garello, K., Verhoef, B. *et al.* SOT-RAM Based Analog in-Memory Computing for DNN Inference. *2020 IEEE Symposium on VLSI Technology* (2020).

[13] Pham, T.-N., Trinh, Q.-K., Chang, I.-J. *et al.* STT-BNN: A Novel STT-MRAM In-Memory Computing Macro for Binary Neural Networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems 12, 2, 569-579* (2022).

[14] Wang, Y., Tang, H., Xie, Y. *et al*. An in-memory computing architecture based on two-dimensional semiconductors for multiply-accumulate operations. *Nature Communications 12, 3347* (2021).

[15] Rzeszut, P., Chęciński, J., Brzozowski, I. *et al*. Multi-state MRAM cells for hardware neuromorphic computing. *Scientific Reports 12, 7178* (2022).

[16] Soliman, T., Chatterjee, S., Laleni, N. *et al*. First demonstration of in-memory computing crossbar using multi-level Cell FeFET. *Nature Communications 14, 6348* (2023).

[17] Das, S., Zaig, A., Schultz, M. *et al.* A four-state magnetic tunnel junction switchable with spin–orbit torques. *Applied Physics Letters 117, 072404* (2020).

[18] Lecun, Y., Bottou, L., Bengio, Y. *et al*. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86, 11, 2278-2324* (1998).