ReRAM Crossbar-based Analog Computing Architecture for Naive Bayesian Engine

Bing Wu, Dan Feng, Wei Tong*, Jingning Liu, Chengning Wang, Wei Zhao, and Mengye Peng
Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System, Engineering Research Center of Data Storage Systems and Technology (School of Computer Science and Technology, Huazhong University of Science and Technology), Ministry of Education of China
{wubin200, dfeng, tongwei, jnliu, chengningwang, weiz, pengmengye}@hust.edu.cn
*Corresponding author

***Abstract*—Recent advances in Resistive RAM (ReRAM) have explored the in-situ Matrix-Vector Multiplication (MVM) ability of crossbar arrays to achieve high energy-efficiency Process-In-Memory (PIM) architectures for Convolutional Neural Network (CNN), image processing, and so on. However, the existing ReRAM-based PIM architectures suffer from considerable additional auxiliary logic and device variations. In this work, we propose a novel analog computing architecture *NB Engine* for classification by implementing Naive Bayesian (NB) algorithm on ReRAM crossbar arrays. The two key steps of the NB algorithm, that is, probability calculation and electing the class that has the highest probability, are elaborately accomplished in our architecture. The ReRAM arrays are both used as storage and computation components. We store the pre-calculated prior probabilities and conditional probabilities of every class in crossbar arrays. Then the probability calculation step is completed in parallel through the MVM operation of the array. In general, the election step is a multiple-comparison procedure and is normally implemented by a comparison tree. Here, we reuse the max pooling module in a conventional CNN PIM architecture to realize a compatible comparison logic. However, neither of the two designs can avoid the overhead of costly high bit-precision Analog-to-Digital Converters (ADCs). So we introduce a novel analog parallel comparison design which does not need any ADCs or other computing logic with better energy-saving and area-efficiency. Our proposed NB Engine is tested by 11 various datasets. The influence of several non-ideal device properties is discussed and the NB Engine exhibits great tolerance to these variations. The experiment results show that our design offers a runtime speedup up to 2289.6× compared with the software-implemented NB classifier with negligible accuracy loss. In addition, the NB Engine saves 96.2% energy consumption and 45.2% array area compared with the CNN PIM compatible design.**

I. Introduction

Conventional von Neumann computation architectures adopt the separate processing units (e.g. CPU) and storage components (e.g. memory and disks). When dealing with massive data and computation, the limitations of the sequential von Neumann architecture are exposed as the data movement between processing units and the memory becomes the major bottleneck of the system causing "memory wall" challenges. The data transfer between CPU and off-chip memory introduces non-negligible latency (around 80ns memory access speed vs. <1ns floating point operation) and huge energy (two orders of magnitude more energy than a floating point operation [1]). Applications like the neural networks put pressure on the current von Neumann system as they are both memory-intensive and computational-intensive.

Recent years, many ReRAM-based Processing-In-Memory (PIM) architectures for convolutional neural networks [2, 3] or other deep learning algorithms [4] are proposed to tackle the "memory wall" challenges. The adjustable resistance ReRAM and its crossbar structure have made it possible to realize both storage and computing in the same array. However, there are still some practical problems in the existing ReRAM-based PIM architectures. The considerable additional auxiliary logic is one of the main challenges. The complexity of the training process in the neural networks such as the gradient descent method makes it costly to be implemented in the PIM architecture as lots of extra computing circuits are needed. Even if only the testing process is concerned, high bit-precision DACs, ADCs, and other auxiliary computation logic are still required that incur non-negligible area and energy overhead [2–6]. Thus, some works aim to reduce the interface overhead by merging the interface [7] or a bitwise model [8].

In this paper, a novel PIM architecture for Naive Bayesian (NB) algorithm, called NB Engine, is proposed to tackle these issues. NB algorithm is widely used to address classification problems due to its good performance, simple structure, and interpretability. The two key steps of the NB algorithm are the probability calculation and the election of a class with the highest probability. First, we elaborately map the probability calculation step of the NB algorithm to the crossbar array and execute it in parallel by the MVM ability of the array. Second, in order to elect the class that has the highest probability, high bit-precision ADCs and additional comparison circuits (e.g. comparison trees) are needed. Instead of using comparison trees for the election operation, we reuse the max-pooling module in the conventional CNN PIM architecture to realize the comparison logic (a CNN PIM compatible design). Moreover, we introduce a novel parallel analog comparison

This work was supported in part by the National Natural Science Foundation of China under Grant 61821003, Grant 61832007, Grant 61772222, and Grant U1705261, in part by the National Science and Technology Major Project No.2017ZX01032-101, and in part by the Fundamental Research Funds for the Central Universities No.2019kfyXMBZ037. This work was also supported by Key Laboratory of Information Storage System, Ministry of Education, China.

978-1-5386-6648-7/19/$31.00 ©2019 IEEE
DOI 10.1109/ICCD46524.2019.00026

design that eliminates the costly ADCs bringing better energy-saving and area-efficient. In our analog comparison design, we use two election modes increasing and binary searching to approximate the result respectively. The proposed architecture exhibits great tolerance to device variations. Sometimes the variations (noises) provide a form of regularization which can help it work better. The sensitivity to the bit-precision and election modes are also studied. The experiment results show that our design offers a runtime speedup up to 2289.6× compared with the software-implemented NB classifier with negligible accuracy loss. In addition, our NB Engine saves 96.2% energy consumption and 45.2% array area compared with the CNN PIM compatible design.

The contributions of this paper are summarized as follows:

- For the first time, we propose a novel analog computing PIM architecture for NB algorithm to get rid of the considerable additional circuits.
- We elaborately map the probability calculation step of the NB algorithm to the crossbar array so that the calculation is completed in one step by the MVM ability of the array.
- We introduce a novel parallel analog comparison design to elect the final class without the costly ADCs.
- The experiment results show the effectiveness of our design from aspects of testing accuracy, performance, energy consumption, and area. Our work also proves that variations (noises) can sometimes make the results better.

II. Preliminary

A. ReRAM Device and Computable Crossbar Array

Figure 1(a) depicts a schematic view of a ReRAM cell. It has a very simple metal-insulator-metal (MIM) structure: a storage layer is sandwiched between the top electrode (TE) and bottom electrode (BE) [9, 10]. A widely accepted filamentary model is used in this paper, in which the switching process of ReRAM is interpreted as the formation or rupture of conductive filaments (CFs) within a cell. The resistance of a ReRAM cell can be programmed to any arbitrary value by applying a programming current with different pulse width and amplitude to the TE & BE. As shown in Figure 1(b), the resistance of a ReRAM cell varies under positive (P) and negative (D) voltage pulses [11].

ReRAM cells can be directly interconnected to each other without transistors, which is called crossbar array. It has a maximum area efficiency of $4F^2$ per cell. Recent studies in ReRAM crossbar array also make it a natural implementation of the matrix-vector multiplication (MVM). Figure 1(c) shows a conceptual example of a MVM in the crossbar array. The current flowing through a ReRAM cell at the wordline i and bitline j is equal to $V_i G_{ij}$. Here, V_i is the voltage applied to the wordline i and G_{ij} is the conduction of the cell. The total current through the bitline j is $\sum_i V_i G_{ij}$, which implements a dot product of $\vec{V} \cdot \vec{G}_j$. Hence, the bitlines together form the result of a MVM $G^T\vec{V}$. The algorithm complexity of MVM is reduced from $O(n^2)$ to $O(1)$. So recent studies take advantage of the MVM ability in the ReRAM array to optimize machine

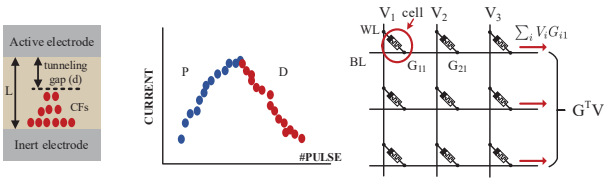


Fig. 1. (a) ReRAM cell. (b) Changeable resistance of the ReRAM cell under positive (P) and negative (D) voltage pulses. (c) Crossbar array and its matrix-vector multiplication (MVM) ability.

learning algorithms [2, 12], image processing [13], function solver [14], and etc.

B. Naive Bayesian Algorithm

For any event A, the conditional probability of A given B is defined by:

$$P(A|B) = \frac{P(AB)}{P(B)}. \tag{1}$$

By deforming the eq.1, we can get the Bayesian theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{2}$$

which gives the relationship between $P(A|B)$ and $P(B|A)$.

1) Naive Bayes: Naive Bayes is a kind of classification algorithm based on the Bayesian theorem. Given a test instance $\boldsymbol{x}$, represented by an attribute value vector $(a_1, a_2, ..., a_m)$. The goal of NB is to find a class label c which has the maximum conditional probability of c given $\boldsymbol{x}$:

$$\hat{c}(\boldsymbol{x}) = \arg\max_{c \in C} \hat{P}(c|\boldsymbol{x}), \tag{3}$$

where C is the collection of all possible class labels c. Then $\hat{c}(\boldsymbol{x})$ is the predicted class label of given $\boldsymbol{x}$.

With eq.2, $\hat{P}(c|\mathbf{x})$ can be rewritten as:

$$\hat{P}(c|\boldsymbol{x}) = \frac{\hat{P}(\boldsymbol{x}|c)\hat{P}(c)}{\hat{P}(\boldsymbol{x})} = \frac{\hat{P}(c)\prod_{i=1}^{m}\hat{P}(a_i|c)}{\hat{P}(\boldsymbol{x})}, \tag{4}$$

where assuming all attributes fully independent of each other given the class. Combining eq.3 with eq.4, we have:

$$\hat{c}(\boldsymbol{x}) = \arg\max_{c \in C} \hat{P}(c) \prod_{i=1}^{m} \hat{P}(a_i|c), \tag{5}$$

as the denominators are all the same and can be omitted.

Thus, before applying Naive Bayes to predict the class label of a given test instance $\boldsymbol{x}$, a training process is needed to obtain the prior probability $\hat{P}(c)$ and conditional probability $\hat{P}(a_k|c)$. They are usually calculated as follows with Laplace smoothing [15].

$$\hat{P}(c) = \frac{\sum_{i=1}^{n}\delta(c_i, c) + \frac{1}{r}}{n+1}, \tag{6}$$

$$\hat{P}(a_k|c) = \frac{\sum_{i=1}^{n}\delta(a_{ik}, a_k)\delta(c_i, c) + \frac{1}{n_k}}{\sum_{i=1}^{n}\delta(c_i, c) + 1}, \tag{7}$$

where n is the number of instances in the training set, r is the number of classes, $\delta(\bullet)$ is a function which outputs an one if its two parameters are identical or a zero otherwise,

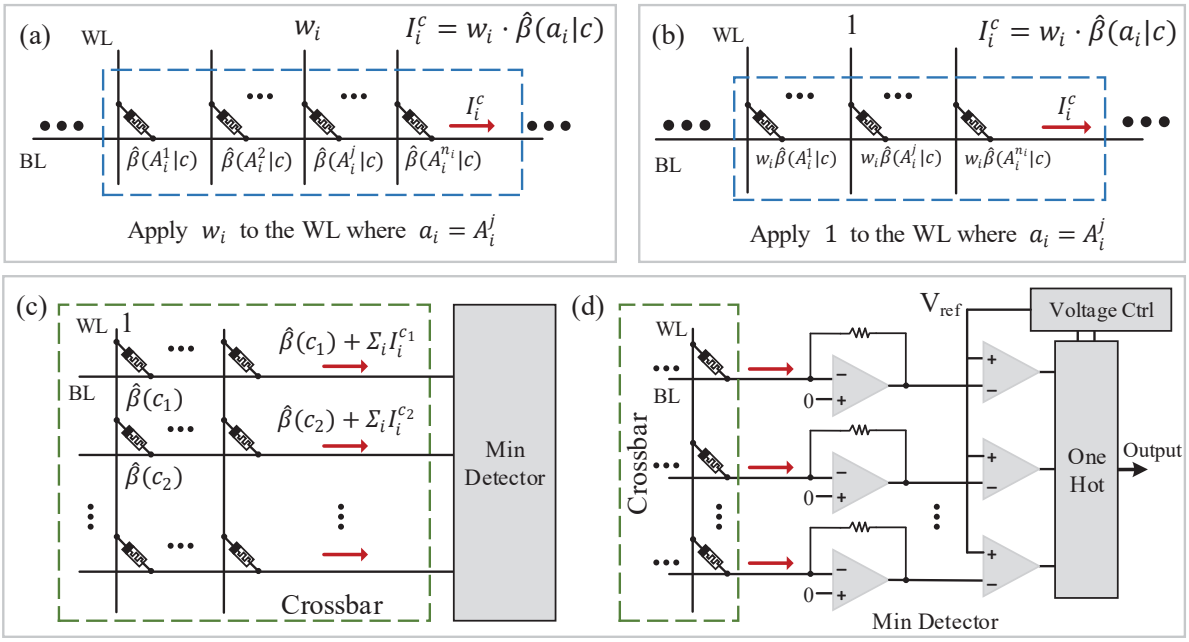


Fig. 2. Naive Bayesian Engine. Diagrammatic processing steps given a test instance $\boldsymbol{x} = (a_1, ..., a_m)$. (a) Calculate ν_i^c represented by current I_i^c. (b) Another way to calculate I_i^c and the weight of ith attribute (w_i) is stored in the cell resistance. (c) Calculate $\varphi(c)$ for every class $c \in C$ by multiple bitlines in the crossbar array. (d) Analog parallel comparison logic to elect the class that has the minimum $\varphi(c)$.

c_i gives the class label of the ith training instance, a_{ik} is the kth attribute value of the ith training instance, and n_k is the number of values for the kth attribute A_k.

As the training process of Naive Bayes is only a simple statistic of the prior probabilities and the conditional probabilities, in this paper, we assume the training has already been done and focus on the testing process.

2) Weighted Naive Bayes: The main weakness of NB classifier is the conditional independence assumption shown in eq.4. It would harm its classification accuracy when the assumption is violated in reality. The state of arts of the NB algorithm [15] assigns every attribute different weight to indicate different importance between each other, which relaxes the conditional independence assumption. The prediction formula is formally defined as:

$$\hat{c}(\boldsymbol{x}) = \arg\max_{c \in C} \hat{P}(c) \prod_{i=1}^{m} \hat{P}(a_i|c)^{w_i}, \tag{8}$$

where w_i is the weight of ith attribute A_i. We can find that the NB classifier is a special case of the Weighted NB (WNB) classifier when all w_i equal to 1. ***In later sections, we will use this more general NB formula.***

III. ReRAM-Based NB Accelerator Design

A. NB Formula Transformation

The testing process of the NB classifier is summarized in eq.8 which cannot be directly applied to the ReRAM crossbar array. So we apply a $log(\bullet)$ operation to eq.8 and then it is rewritten as:

$$\hat{c}(\boldsymbol{x}) = \arg\max_{c \in C} \{\hat{\rho}(c) + \sum_{i=1}^{m} w_i \cdot \hat{\rho}(a_i|c)\}, \tag{9}$$

where $\hat{\rho}(\bullet)$ denotes $log\hat{P}(\bullet)$.

Let $\hat{q}(c)$ refer to $\hat{\rho}(c) + \sum_{i=1}^{m} w_i\hat{\rho}(a_i|c)$. We can write the calculation of $\hat{q}(c)$ in the form of a dot product $\vec{v} \cdot \vec{g}$, where $\vec{v} = [1\ w_1\ ...\ w_m]$ and $\vec{g} = [\hat{\rho}(c)\ \hat{\rho}(a_1|c)\ ...\ \hat{\rho}(a_m|c)]$. So it is possible to calculate $\hat{q}(c)$ of every class by the MVM ability of the crossbar array. Detail steps are shown in section III-B.

Another problem is that the value of $\hat{\rho}(\bullet)$ is less than zero because $\hat{P}(\bullet) \in (0, 1)$. Negative numbers cannot be represented by the conductance of ReRAM cells as the conductance is always positive. Thus, the final formula is transformed to:

$$\hat{c}(\boldsymbol{x}) = \arg\min_{c \in C} \{\hat{\beta}(c) + \sum_{i=1}^{m} w_i \cdot \hat{\beta}(a_i|c)\}, \tag{10}$$

where $\hat{\beta}(\bullet)$ denotes $-log\hat{P}(\bullet)$ and is always positive. The $-log(\bullet)$ operation can be directly added to the training process. After training, we will store all the prior probabilities and the conditional probabilities in the array. ***For convenience, we make following abbreviations:***

$$\nu_i^c \equiv w_i \cdot \hat{\beta}(a_i|c), \quad \varphi(c) \equiv \hat{\beta}(c) + \sum_{i=1}^{m} \nu_i^c$$

B. Mapping NB to the Crossbar Array

Generally, the NB classifier (eq.10) is divided into two steps:

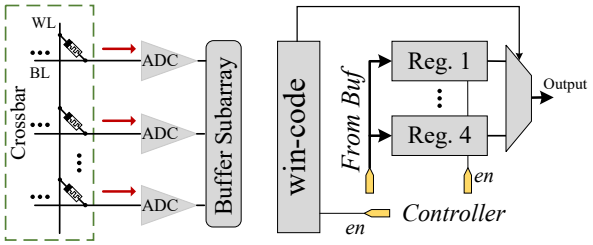


Fig. 3. Conventional CNN PIM Compatible Design. Implement Min Detector by the existing max pooling module.

- Calculate $\varphi(c)$ of belonging to every class where $c \in C$.
- Elect the class c that has the minimum $\varphi(c)$.

The crossbar-based hardware implementation of NB is shown from these two aspects.

1) Calculate $\varphi(c)$: Given the class c and test instance $\boldsymbol{x} = (a_1, ..., a_m)$, the calculation of $\varphi(c)$ is a dot product operation represented by $[1\ w_1\ ...\ w_m] \cdot [\hat{\beta}(c)\ \hat{\beta}(a_1|c)\ ...\ \hat{\beta}(a_m|c)]$. Our goal is to map this dot product operation to the crossbar array.

After training, we store every prior probability $\hat{\beta}(c)$ and every conditional probability $\hat{\beta}(A_i^j|c)$ in the crossbar array in the form of ReRAM conductance, where $c \in C$ and A_i^j refers to the jth value of the ith attribute (A_i). Note that a_i is the ith attribute value for the instance $\boldsymbol{x}$ and it belongs to A_i. This means there must be a j that meets $A_i^j = a_i$. Thus we can compute the result of $\varphi(c)$ in the way shown in Figure 2(a). For ith attribute A_i, voltage w_i is applied only to the wordline that a_i equals to the A_i^j. The other wordlines belonging to A_i are grounded. The current gathered on this sub-bitline (I_c^i) form the result of ν_c^i. With the addition of $\hat{\beta}(c)$, the final result of $\varphi(c)$ is obtained as current on one bitline. As shown in Figure 2(c), multiple bitlines together give answers $\varphi(c)$ to all classes.

We also propose an optimization to the input voltage. Because analog input voltage (i.e. w_i) might result in inaccuracy due to the I-V nonlinearity of the ReRAM cell. The weight w_i is included in the cell conductance shown in Figure 2(b). Only 1-bit input voltages are required.

2) Elect the Minimum $\varphi(c)$: To detect the minimum $\varphi(c)$ and its source class (Min Detector module in Figure 2(c)), usually a comparison tree design is needed. By reusing the existing max pooling module in a conventional CNN PIM architecture (e.g. PRIME [2]), we first introduce a **compatible design** to implement Min Detector. As shown in Figure 3, 4:1 max pooling hardware is adopted and n:1 max pooling can be achieved by multiple steps (for $n > 4$). We use the same logic to elect the minimum $\varphi(c)$ with its source class. First, the results of $\varphi(c)$ for each $c \in C$ are stored in the buffer after the conversion of ADCs. Second, four inputs $\{o_i\}$ from buffer are stored in the registers, $i = 1, 2, 3, 4$. Then, we use ReRAM array to execute the dot products between $\{o_i\}$ and six sets of weights [1,-1,0,0], [1,0,-1,0], [1,0,0,-1], [0,1,-1,0], [0,1,0,-1], [0,0,1,-1]. So the sign results of $\boldsymbol{a} = \{(o_i - o_j), i \neq j\}$ are got and then send to Winner Code register (win-code). At last, according to the code, the maximum (and its source) can

Algorithm 1: Min Detector

Input: Output of every bitline represented by $\vec{V}_o$
Output: Minimum $\varphi(c)$ represented by V_{Min} and its source class label c

if *Election_Mode* = *Inc_Voltage* **then**
  $V_{ref} \leftarrow V_{lowest}$
  $\vec{r} \leftarrow compare(V_{ref}, \vec{V}_o)$
  while $\vec{r} = \vec{0}$ **do**
    // $\tilde{1}$ refers to the minimum voltage step
    $V_{ref} \leftarrow V_{ref} + \tilde{1}$
    $\vec{r} \leftarrow compare(V_{ref}, \vec{V}_o)$
else `/* Election_Mode is Binary_Search */`
  $V_l \leftarrow V_{lowest}$, $V_r \leftarrow V_{highest}$
  while $V_l \leq V_r$ **do**
    $V_{mid} \leftarrow (V_l + V_r) >> 1$
    $\vec{r} \leftarrow compare(V_{mid}, \vec{V}_o)$
    if $is\_OneHot(\vec{r})$ **then**
      $V_{ref} \leftarrow V_{mid}$
      break
    else if $\vec{r} = \vec{0}$ **then**
      $V_l \leftarrow V_{mid} + \tilde{1}$
    else
      $V_r \leftarrow V_{mid} - \tilde{1}$
$V_{Min} \leftarrow V_{ref}$
// find the source class label according to the code $\vec{r}$
$c \leftarrow source(\vec{r})$

be determined by the hardware thus the max pooling logic is implemented. Here we are going to detect the minimum $\varphi(c)$, then we reverse $\boldsymbol{a}$ and feed it into win-code so that the output would be the minimum. By reusing this module, the winner of the previous time is sent to the comparison again until the final winner (the minimum $\varphi(c)$) is produced. As a result, electing the minimum $\varphi(c)$ among n classes requires $\lceil \frac{n-1}{3} \rceil$ comparison steps.

Furthermore, the state of art PIM architecture [2, 6] points out that ADCs are costly in both energy and area. To find the minimum $\varphi(c)$ and elect its source class, high bit-precision ADCs are needed in traditional comparison tree design or our proposed compatible design. Here, we propose an analog parallel comparison design eliminating the costly ADCs. In the conventional ReRAM array, a (or multiple) fixed reference voltage(s) V_{ref} is (are) used during a read operation to determine the binary (multi-level) logical value stored in the cell. We introduce an additional voltage controller as shown in Figure 2(d). The operational amplifier connected to the each bitline outputs a voltage that represents the value of $\varphi(c)$. Then the voltage controller generates a dynamically varying threshold V_{ref}, and V_{ref} is shared by all the voltage comparators. V_{ref} keeps rising from the lowest voltage. When V_{ref} is larger than the output voltage of any one operational amplifier, the corresponding voltage comparator generates a

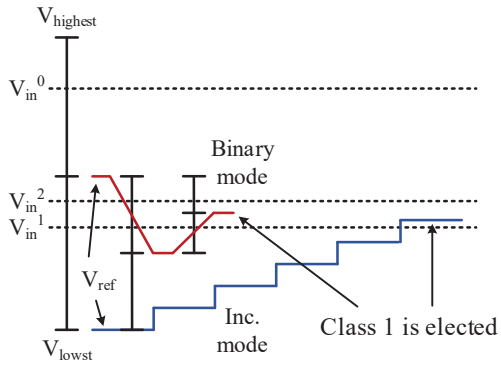


Fig. 4. Comparison steps for the increasing mode (Inc. mode) and binary searching mode (Binary mode).

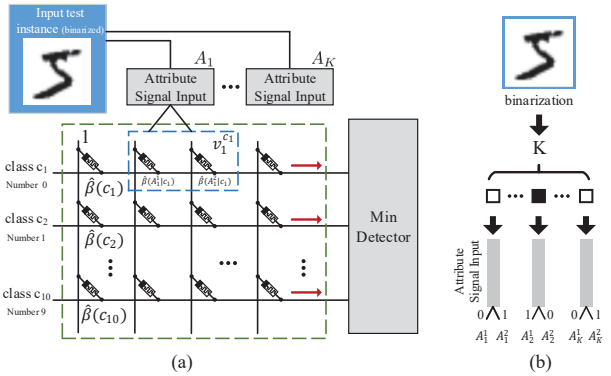


Fig. 5. (a) Case study of MNIST on ReRAM crossbar array. (b) Attribute signal input to every attribute group A_i, $K = 784$.

high-level output. At the same time, V_{ref} stops rising and keeps its value so that only one voltage comparator generates a high-level output. The outputs of all voltage comparators form an one-hot code. Then the One-Hot module parses out the corresponding source class that represents the minimum $\varphi(c)$. Instead of using an increasing voltage in the controller, we also propose a binary searching scheme to faster approach the result. The pseudo-code for the two strategies (increasing mode and binary searching mode) is shown in Algo. 1. A DAC is needed to generate the reference voltage (V_{ref}). V_{lowst} and $V_{highest}$ refer to the lowest and highest voltage levels and are initially stored in the controller calculated by assuming all cells in HRS or LRS. $\tilde{1}$ refers to the minimum voltage step. Function $compare$ is equivalent to the multiple voltage comparators in Figure 2(d). Function $is\_OneHot(\vec{r})$ judges whether the input $\vec{r}$ is one-hot code or not. And function $source(\vec{r})$ parses out the source class according to the $\vec{r}$, which is the One-Hot module in Figure 2(d).

The basic comparison step for the two searching mode is depicted in Figure 4. By reducing the search range by half each time, the binary searching mode has fewer comparisons in most cases. The reason why we haven't given up the voltage increasing mode is that it is simpler and may have fewer comparisons when the minimum $\varphi(c)$ approaches V_{lowst}. We also verify it in the performance evaluation.

C. Case Study of MNIST

The MNIST [16] dataset consists of handwritten digit images. The goal is to distinguish the digit value of every image. Every digit image is normalized and centered in a fixed size of 28×28 pixels. Each pixel is represented by a value between 0 and 255, where 0 means white and 255 means black. First, we binarize the value of each pixel to 0 (≤ 127) and 1 (> 127). So there are 784 (28×28) attributes and each attribute only consists of two values (0 or 1). The handwritten digits range from 0 to 9 so that we have 10 classes $c_1, ..., c_{10}$ corresponding to $[0, 9]$. After training, we store the value of $\hat{\beta}(c_r)$ and $\hat{\beta}(A_i^j|c_r)$ in the form of cell conduction in the array as shown in Figure 5(a), where $r \in [1, 10]$ (10 classes), $i \in [1, 784]$ (784 attributes), and $j \in [1, 2]$ (2 attribute values for each attribute). Here, we take the ordinary NB algorithm as an example, which means $w_i = 1$ for all i. In the testing process, we have a test instance $\boldsymbol{x} = (a_1, ..., a_{784})$ where $a_i \in A_i$. The input voltage is only applied to the wordline meeting $a_i = A_i^j$ (recall Figure 2(b)). So for each attribute, the input voltages form an one-hot coded value. As shown in Figure 5(b), the binarized pixel value are one-hot coded and then input to the array in the form of voltage. Thus, the NB algorithm of running the MNIST benchmark is achieved on crossbar arrays. Note that if there are too many attribute values, we can store them separately in multiple arrays.

IV. Evaluation

A. Benchmarks

We run our experiments on a collection of 11 benchmarks including MNIST [16] which is a classical pattern recognition dataset of handwritten digits. Other benchmarks are selected from the University of California at Irvine (UCI) repository [17]. They are all classical classification problems that belong to a wide range of domains with different data characteristics. The detailed information of these datasets is shown in Table I. In some datasets, e.g. anneal, there are missing attribute values. The missing attribute value is replaced with the mode of a nominal attribute value if the attribute is discrete or with the mean of numerical attribute values from the available data if the attribute is numerical. As the Naive Bayesian algorithm requires discrete attribute values, numerical values are discretized using the MDLP cut proposed by [18]. The prior probabilities $\hat{P}(c)$ and the conditional probabilities $\hat{P}(a_k|c)$ of all classes are pre-calculated by the same eqs. 6 and 7 for every benchmark.

Note that, for those datasets that do not give separated training and testing set, we introduce the shuffling algorithm to upset the dataset and then separate it into two parts for training and testing respectively.

B. Experiment Setup

In our experiments, we compare our NB Engine with a CPU-based software implementation of NB from aspects of accuracy and runtime. We also compare the NB Engine with the CNN PIM compatible design to show the energy and area efficiency of our design. The software-implemented NB

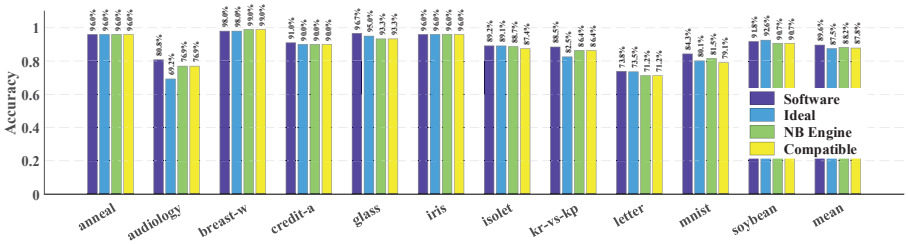


Fig. 6. Test accuracy results under the different implementation of NB.

TABLE I
DATASET DESCIPTION

Dataset	#Instance	#Attribute	#Class	M.-v.[1]	N.-v.[2]
anneal	898	38	6	Y	Y
audiology	226	69	24	Y	N
breast-w	699	10	2	Y	N
credit-a	690	15	2	Y	Y
glass	214	10	7	N	Y
iris	150	4	3	N	Y
isolet	7797	617	26	N	Y
kr-vs-kp	3196	37	2	N	N
letter	20000	16	26	N	Y
mnist	60000	784	10	N	N
soybean	683	35	19	Y	N

[1] Missing-value : Missing attribute values.
[2] Numeric-value : Numeric attribute values.

algorithm is evaluated in the GEM5 [19] with the NVMain [20] simulator[1] by running the benchmark datasets mentioned before. Table II shows the system configuration in the simulation.

TABLE II
SIMULATION PARAMETERS

Parameter	Value
CPU	4 cores, out of order, 3GHz
L1 I&D-Cache	Private, 16KB I-cache, 16KB D-cache, 2-way, 2 cycles access
L2 Cache	Shared, 4MB, 16-way, 20 cycles access, 64-byte cache line
DRAM Memory	4GB, DDR3-1333, 2 channels, 2 rank per channel, 8 banks per rank, tRCD-tCL-tWR-tWTR-tCWD 9-10-10-5-7

To evaluate our crossbar-based NB Engine, we modify the NeuroSim [21] which is an integrated simulation framework for benchmarking synaptic devices and array architecture. Our test parameters are set as follows. We choose 32nm technology node for the design and adopt an ideal synaptic ReRAM device and a Ag:a-Si stacked ReRAM device proposed by [11] to evaluate the NB engine respectively. The parameters of the two devices are shown in Table III. The nonlinear weight update [22], C2C device variation [23], and interconnect IR-drop [24, 25] are included in the simulation for the real device (Ag:a-Si). The details of these non-ideal parameters can be found in NeuroSim. The ADC, DAC, and comparator we refer to are shown in [26]. For simplicity, our experiments only consider the ordinary NB algorithm which assumes $w_i = 1$.

[1]The GEM5 and NVMain simulator form an effective full-system simulation to evaluate the program.

But remember, the improved NB algorithm (WNB) has been considered in our design.

TABLE III
DEVICE PARAMETERS

	Ag:a-Si	Ideal Device
# of conductance states	97	97
Nonlinearity (weight inc./dec.)	2.4/-4.88	0/0
R_{ON}	$26M\Omega$	$26M\Omega$
ON/OFF ratio	12.5	12.5
Cycle-to-Cycle variation (σ)	3.5%	0%
Interconnect IR-drop	YES	NO

C. Classification Accuracy Results

1) Test Accuracy: For the evaluation purpose, here we compare the test accuracy of the NB Engine with the software-implemented NB classifier. We test NB Engines in two configurations as mentioned before, one using the real ReRAM device and the other using the ideal ReRAM device, which is abbreviated as 'NB Engine' and 'Ideal' in the figure respectively. The voltage controllers in the two configurations are all set with the 8-bit precision DAC. In addition, the CNN PIM compatible design is another comparison reference (abbreviated as 'Compatible' in the figure) which is configured with 8-bit precision ADCs based on the real device.

Figure 6 depicts the test accuracy results under the different implementations of the NB algorithm by running benchmarks described earlier. In most benchmarks, software-implemented NB classifier achieves highest test accuracy and it gets an average accuracy of 89.6%. In all benchmarks except for Soybean, we find that the NB Engine based on ideal devices always has lower accuracy than software implementation with a maximum degradation of 11.6% in audiology. In Soybean, the improvement of the accuracy might come from the generalization ability of the low bit-precision values (compared with the float point values in the software implementation). But in most cases, lower bit-precision values result in lower test accuracy.

Our NB Engine based on the real device achieves an average test accuracy of 88.2% which is only worse than the software implementation with negligible accuracy degradation of -1.4%. This demonstrates the tolerance of the NB Engine to the variations. In some benchmarks, e.g. audiology and kr-vs-kp, the accuracy of the NB Engine based on the ideal device decreases dramatically, whereas the NB Engine based on the

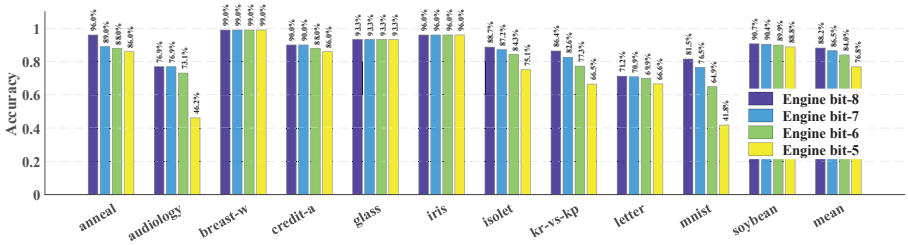


Fig. 7. Test accuracy results under different bit-precision NB Engine.

real device does not. The noises of C2C variations, nonlinear weight update, and etc. provide a form of regularization helping to generalize better, which is also proved in software machine learning algorithms [27, 28].

The results of the CNN PIM compatible design are similar to our real-device-based NB Engine with a little accuracy degradation in MNIST (-2.4%). The possible reason might be the problem of bias accumulation after the numerical conversion of ADCs. We require several arrays to complete the NB algorithm for MNIST. After the conversion of ADCs, the value actually has some biases. Then the accumulations between the results from ADCs of every array cause some deviations in the final outputs which degrades the accuracy.

2) The sensitivity to the bit-precision of the voltage controller: For the NB Engine, we vary the bit-precision of the DAC in the voltage controller to study its sensitivity. The NB Engine with different bit-precision DAC is abbreviated as 'Engine bit-n' in the figure, e.g. Engine bit-8. The test results are in good agreement with common sense. As the bit-precision of the voltage controller decreases, the test accuracy of benchmarks decreases as well. In addition, different benchmarks have different tolerances to the bit-precision of the voltage controller. With the decrease of the bit-precision, the NB Engine remains a stable test accuracy in some benchmarks such as breast-w, while suffers from considerable accuracy loss in some other benchmarks such as Audiology. **Note that** the results of the sensitivity analysis here are not aiming to use a low bit-precision DAC in the voltage controller. The bit-precision of the DAC is fixed to 8 to satisfy all kinds of applications. But we can choose a larger searching pace for Min Detector according to the benchmark characteristics if the benchmark is tolerant to the bit-precision. A larger searching pace each time of Min Detector can further improve test speed as fewer comparisons are needed. Moreover, because the voltage controller is shared by the whole array (even multiple arrays), one 8-bit DAC does not cause large area overhead. In later evaluations, the bit-precision of the DAC in the voltage controller is set to 8-bit if no otherwise specified.

D. Performance Results

1) Speedup normalized to the software implementation: Figure 8 depicts the normalized runtime speedup with respect to the software-implemented NB classifier. The results show that our NB Engine gains a up to 2289.6× runtime speedup. A performance improvement of 235.2×, 2289.6×, 65.1×, 1057.0×, and 110.3× is gained in Audiology, Isolet, Letter, MNIST, and Soybean respectively. But in other benchmarks, there is only a little improvement. It can be found that the more attributes and classes that a benchmark has (see figure I), the more performance improvements can NB Engine gain. This is because when there are only a few attributes[2] and classes, the advantages of parallel MVM operations in the crossbar array will not bring significant performance improvements as the scale of the MVM operation is too small.

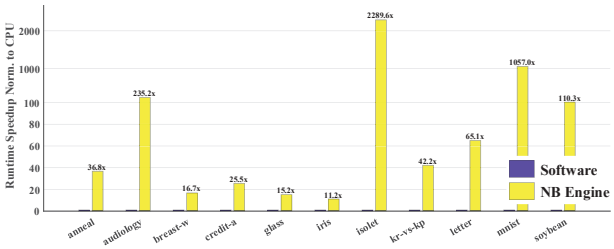


Fig. 8. Runtime Speedup normalized to CPU.

2) The sensitivity to the bit-precision and searching mode of the voltage controller: Figure 9 shows the influence of the bit-precision and searching mode in the NB Engine. The results are normalized to the 8-bit precision voltage controller with binary searching mode. In the figure, inc. is the abbreviation for the voltage increasing mode and bit-n represents that the n-bit precision is used. Here, we show four representative benchmarks. As mentioned earlier, the bit-precision of the voltage controller is fixed to 8. The change of the bit-precision here only represents a larger searching pace each time.

The results show that a lower bit-precision DAC in the voltage controller leads to fewer comparisons for NB Engine in both two searching modes. In most benchmarks (not shown here), the results are similar to that in Isolet, which validates the effectiveness of the binary searching mode. But in some benchmarks, the NB Engine performs better using voltage increasing mode with a lower bit-precision DAC (e.g. 5-bit precision mode in Audiology, Glass, and MNIST). The reason is that a smaller result value is tended to be produced in these benchmarks. So the voltage increasing mode which starts comparisons from the lowest voltage value has fewer comparisons than the binary searching mode which is basically stable. Thus, we can use a low-precision DAC with voltage increasing mode to accomplish better performance in those benchmarks if the test accuracy is also acceptable (e.g. Glass

[2]More specifically, we refer to the number of attribute values

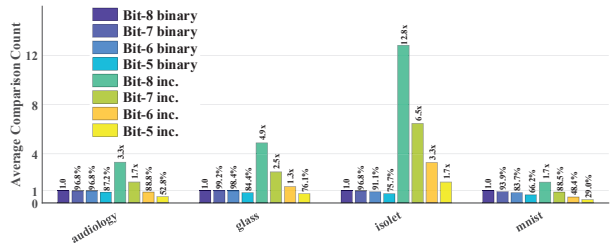
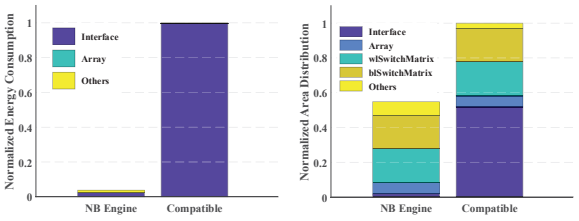


Fig. 9. Average number of comparisons in NB Engines under different bit-precision DAC and different searching mode. The results are normalized to the 8-bit precision DAC with the binary searching mode.



(a) (b)

Fig. 10. Energy consumption (a) and Area distribution (b) normalized to the CNN PIM compatible design.

in the 5-bit precision mode or Audiology in the 6-bit precision mode).

E. Energy & Area Results

We take MNIST benchmark as an example to show the energy and area efficiency of our design. In the CNN PIM compatible design, the number of ADC is set to 10 as MNIST has 10 classes. The number of arrays meets the need of MNIST and the ADCs are shared by these arrays. We don't set 10 ADCs per array as the ADC is too costly in both energy and area. Later tests validate that even if there are only 10 ADCs, they still dominate the energy consumption and area.

Figure 10 shows the energy consumption per test instance of NB Engine compared with the CNN PIM compatible design. In the compatible design, the ADCs dominate nearly 100% energy consumption. That is because the ReRAM cell we choose has a high resistance ($R_{on} = 26M\Omega$) so that the arrays only consume little energy. Our NB Engine consumes only 3.8% energy (96.2% saved) with respect to the CNN PIM compatible design. On the one hand, the elimination of high bit-precision ADCs used in the conventional PIM architecture results in energy saving. On the other hand, multiple steps (longer processing time) are needed in the compatible design as ADCs are shared by arrays. The NB Engine processes a test instance with a shorter time that further reduces energy consumption. Increasing the number of ADCs to reduce processing time in the compatible design cannot solve the problem as the total power consumption of ADCs increases proportionately.

The area distribution is shown in Figure 10. A wl/blSwitch-Matrix is adopted for fully parallel voltage input to the array rows or columns, which is specially designed for matrix-vector multiplication [21]. Our NB Engine saves 45.2% area compared with the CNN PIM compatible design. The area overhead of the interface ratio decreases from 51.9% to 1.8%.

V. Conclusion

In this paper, we build a novel analog computing architecture for the NB algorithm based on ReRAM crossbar arrays. The special design of the probability calculation by the MVM ability of the ReRAM crossbar array is well discussed. Moreover, we propose a novel parallel comparison design without high bit-precision ADCs to elect the source class (the final result). The robustness of our design under device variations is also demonstrated. The NB Engine shows negligible accuracy loss or even improvements in some benchmarks due to a form of regularization provided by the variations (noises). A up to 2289.6× runtime speedup is achieved in the NB Engine compared with the software implementation. Besides, we analyze the performance factors including benchmark characteristics (the number of attributes and classes), bit-precision of the voltage controller, and the election mode. In addition, the NB Engine saves 96.2% energy consumption and 45.2% area compared with the CNN PIM compatible design.

References

[1] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "Gpus and the future of parallel computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, 2011.

[2] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 27–39.

[3] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[4] F. Chen, L. Song, H. H. Li, and Y. Chen, "Zara: A novel zero-free dataflow accelerator for generative adversarial networks in 3d reram," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, p. 133.

[5] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Binary convolutional neural network on rram," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 782–787.

[6] Z. Zhu, J. Lin, M. Cheng, L. Xia, H. Sun, X. Chen, Y. Wang, and H. Yang, "Mixed size crossbar based rram cnn accelerator with overlapped mapping method," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[7] B. Li, L. Xia, P. Gu, Y. Wang, and H. Yang, "Merging the interface: Power, area and accuracy co-optimization for rram crossbar-based mixed-signal computing system," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 13.

[8] L. Ni, Z. Liu, W. Song, J. J. Yang, H. Yu, K. Wang, and Y. Wang, "An energy-efficient and high-throughput bitwise cnn on sneak-path-free digital reram crossbar," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2017, pp. 1–6.

[9] B. Wu, D. Feng, W. Tong, J. Liu, S. Li, M. Yang, C. Wang, and Y. Zhang, "Aliens: a novel hybrid architecture for resistive random-access memory," in *2018 IEEE/ACM International*

Conference on Computer-Aided Design (ICCAD). IEEE, 2018, pp. 1–8.
[10] C. Wang, D. Feng, W. Tong, J. Liu, B. Wu, W. Zhao, and Y. Zhang, "Design and analysis of address-adaptive read reference settings for multilevel cell cross-point memory arrays," *IEEE Trans. Electron Devices*, 2019.
[11] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano letters*, vol. 10, no. 4, pp. 1297–1301, 2010.
[12] Y. Jiang, J. Kang, and X. Wang, "Rram-based parallel computing architecture using k-nearest neighbor classification for pattern recognition," *Scientific reports*, vol. 7, p. 45233, 2017.
[13] C. Li, M. Hu, Y. Li, H. Jiang, N. Ge, E. Montgomery, J. Zhang, W. Song, N. Dávila, C. E. Graves *et al.*, "Analogue signal and image processing with large memristor crossbars," *Nature Electronics*, vol. 1, no. 1, p. 52, 2018.
[14] M. Le Gallo, A. Sebastian, R. Mathis, M. Manica, H. Giefers, T. Tuma, C. Bekas, A. Curioni, and E. Eleftheriou, "Mixed-precision in-memory computing," *Nature Electronics*, vol. 1, no. 4, p. 246, 2018.
[15] L. Jiang, L. Zhang, L. Yu, and D. Wang, "Class-specific attribute weighted naive bayes," *Pattern Recognition*, vol. 88, pp. 321–330, 2019.
[16] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[17] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[18] U. Fayyad and K. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," 1993.
[19] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.
[20] M. Poremba and Y. Xie, "Nvmain: An architectural-level main memory simulator for emerging non-volatile memories," in *2012 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 2012, pp. 392–397.
[21] P.-Y. Chen, X. Peng, and S. Yu, "Neurosim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures," in *2017 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2017, pp. 6–1.
[22] P.-Y. Chen and S. Yu, "Technological benchmark of analog synaptic devices for neuro-inspired architectures," *IEEE Design & Test*, 2018.
[23] C. Wang, D. Feng, W. Tong, J. Liu, Z. Li, J. Chang, Y. Zhang, B. Wu, J. Xu, W. Zhao *et al.*, "Cross-point resistive memory: Nonideal properties and solutions," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 24, no. 4, p. 46, 2019.
[24] C. Wang, D. Feng, J. Liu, W. Tong, B. Wu, and Y. Zhang, "Daws: Exploiting crossbar characteristics for improving write performance of high density resistive memory," in *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 281–288.
[25] B. Wu, D. Feng, W. Tong, J. Liu, C. Wang, W. Zhao, and Y. Zhang, "A low power reconfigurable memory architecture for complementary resistive switches," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.
[26] L. Kull, T. Toifl, M. Schmatz, P. A. Francese, C. Menolfi, M. Braendli, M. Kossel, T. Morf, T. M. Andersen, and Y. Leblebici, "A 3.1 mw 8b 1.2 gs/s single-channel asynchronous sar adc with alternate comparators for enhanced speed in 32 nm digital soi cmos," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 12, pp. 3049–3058, 2013.
[27] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.
[28] A. Graves, "Practical variational inference for neural networks," in *Advances in neural information processing systems*, 2011, pp. 2348–2356.