IEEE Journal on Exploratory Solid-State Computational Devices and Circuits

Received 22 February 2022; revised 19 April 2022, 6 May 2022, and 17 May 2022; accepted 20 May 2022.
Date of publication 23 May 2022; date of current version 9 June 2022.

Digital Object Identifier 10.1109/JXCDC.2022.3177588

An Algorithm-Hardware Co-Design for Bayesian Neural Network Utilizing SOT-MRAM's Inherent Stochasticity

ANNI LU, YANDONG LUO, and SHIMENG YU (Senior Member, IEEE)

Georgia Institute of Technology, Atlanta, GA 30332 USA

CORRESPONDING AUTHOR: S. YU (shimeng.yu@ece.gatech.edu)

This work was supported in part by Intel.

ABSTRACT Probabilistic machine learning plays a central role in the domains such as decision-making and autonomous control benefitting from its ability of representing and manipulating uncertainty about models and predictions. Until now, there are few hardware considerations to address the intensive computation and true random number generation for Bayesian neural network (BayesNN), whose weights are represented by probability distributions. In this article, we propose to apply the local reparameterization trick to alleviate the burden of random number generators (RNGs), which could be implemented by utilizing the inherent random noise of spin-orbit torque magnetic random access memory (SOT-MRAM). Sampling strategies are discussed to significantly reduce the number of operations and parameters of BayesNN. A device-circuit-system benchmark framework is then developed to evaluate the effects of device nonidealities such as the bias and variation of switching probability. The evaluation on the CIFAR-10 dataset suggests that BayesNN could achieve comparable accuracy as conventional deep neural network (DNN) with acceptable hardware overhead but provide much better uncertainty calibration with respect to out-of-distribution (OOD) inputs (rotated images as the example).

INDEX TERMS Bayesian neural network (BayesNN), magnetic tunnel junction (MTJ), neural network hardware accelerator, probabilistic computing, random number generation.

I. INTRODUCTION

BAYESIAN deep learning has recently attracted more attention due to its applications such as medical diagnosis, recommendation systems, nonlinear dynamic system control, and attack detection [1]. Compared with conventional deep neural network (DNN), the parameters of Bayesian neural network (BayesNN) are modeled as probability distributions, which offers principled uncertainty estimates instead of just point estimates in a standard network. The prior probabilities of parameters are usually modeled by Gaussian processes characterized by a mean and variance. However, the repeated sampling from parameter distributions causes even more expensive computation and extra overhead of random number generation, thereby limiting its deployment on edge devices with constrained resources.

Compute-in-memory (CIM) is a promising solution to alleviate the memory access bottleneck of neural networks by configuring the memory arrays for dot-product computation in a parallel fashion by summing up the column currents [2]. The CIM scheme can be expanded to BayesNN with the Gaussian-shaped weights as the inherent stochasticity of certain memory devices can be exploited to generate Gaussian random numbers, e.g., the random power-on state of SRAM [3], the switching probability of magnetic tunnel junction (MTJ) [4], the thermal noise in resistive random access memory (RRAM) during reading or programming [5], [6], and the programming uncertainty in ferroelectric field-effect transistor (FeFET) [7].

Until now, the hardware design of BayesNN is still at an early stage compared with that for a variety of DNNs. Few examples of BayesNN hardware implementations are briefed as follows. VIBNN [8] is an FPGA implementation with an energy-efficient CMOS design of the RNG. Work [9] is also a CMOS design that proposed to reduce the computation through feature decomposition and memorization. Work [10] is another FPGA-based accelerator for BayesNN with

Monte Carlo dropout method. Despite these improvements, the projected hardware costs for a CMOS implementation still could be expensive. Alternatively, works [5], [6], [11] proposed CIM-based designs with RRAM or MTJ-based probabilistic array (p-array), and however, the preprocessing of [5] is too complex, especially for large-scale models, and works [6], [11] managed to reduce the burden of RNG, while the effectiveness of the uncertainty estimation might be sacrificed. Besides, there are no full-stack or end-to-end tools from device to circuit, architecture, and algorithm to evaluate the overall system-level software and hardware performance for BayesNN design space exploration.

In this article, a BayesNN accelerator is designed with software–hardware co-optimization and a device-circuit-algorithm benchmark framework is developed. The preliminaries of BayesNN software baseline and the limitations of prior hardware designs are discussed in detail in Section II. In Section III, the local reparameterization trick is proposed to be applied for the first time on BayesNN hardware accelerators to effectively reduce the burden of RNG, with the multiply–accumulate (MAC) operations still processed through conventional crossbar arrays and only vectors of Gaussian random numbers generated through probabilistic crossbar arrays. In Section IV, the sampling strategies are discussed. Considering tradeoffs between accuracy, uncertainty calibration, and hardware costs, taking about ten samples of only last several layers with a larger Gaussian variance is desired. In Section V, we simulated the effects of device nonidealities such as the bias and variation of switching probability and benchmarked the hardware performance in array level and system level, respectively.

II. PRELIMINARIES

A. BAYESIAN NEURAL NETWORK (BayesNN)

Typical classification models are usually judged on accuracy; however, the calibration of probabilistic predictions is also important for many decision-making applications such as autonomous vehicles. While traditional neural networks usually tend to overfit classification accuracy when minimizing the cross-entropy loss, BayesNN is able to capture and calibrate uncertainty of predictions and models by introducing probability distribution over the weights. Compared with conventional DNN, BayesNN can effectively resolve overfitting, avoid erroneous and overconfident predictions on abnormal inputs, and therefore can be used for out-of-distribution (OOD) detection (e.g., for rotated input images) and of critical importance for real-world applications.

We measured some metrics of BayesNN with VGG-16 topology on the CIFAR-10 dataset using floating-point computations as the software baseline. The classical variational approach "Bayes by Backprop" [12] is utilized. The inference of BayesNN is simply the vector-matrix multiplication between input and weight similar to the conventional neural networks [2], while the elements of the weights are Gaussian random numbers. The computation can be represented as $\sum_i x_i N(\mu_{i,j}, \sigma_{i,j})$. A comparable classification accuracy of 89.52% as in the conventional DNNs is obtained and used as the software baseline without nonideal hardware effects. Various rotation degrees are applied on the CIFAR-10 images as the OOD inputs. The BayesNN is able to capture more uncertainty as shown from the higher average predicted entropy (PE) in Fig. 1(a). Expected calibration error (ECE) is a metric to compare the model output pseudo-probabilities (confidence) to model accuracies, and the reliability diagram could visually show the model calibration. In Fig. 1(b), BayesNN shows much better calibration ability with the lower ECE regardless of the rotation degree. In the reliability diagram of Fig. 1(c), BayesNN is closer to the linear dashed line where the model confidence of classification results exactly matches the actual accuracy in an ideal situation for in-distribution (ID) inputs. To measure the ability of OOD detection, receiver operating characteristic (ROC) curve is plotted in Fig. 1(d) showing the true positive rate (TPR) against the false positive rate (FPR) of the binary OOD classification with varying PE as the detection threshold. The BayesNN performs better shown from the higher area under ROC (AUROC) metric.

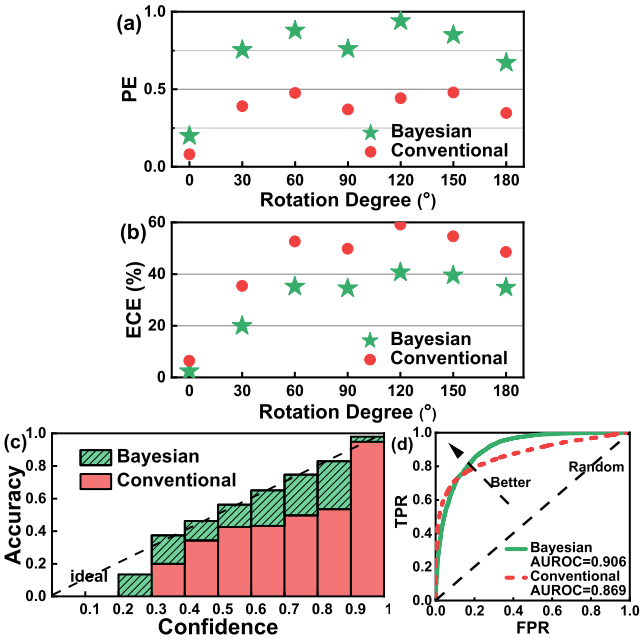

FIGURE 1. Baseline performance of Bayesian neural network VGG16 on the Cifar-10 dataset. (a) Average PE and (b) ECE of BayesNN and conventional DNN in terms of rotation degree of CIFAR-10 images as OOD inputs. (c) Reliability diagram of Bayesian and conventional neural networks of regular ID inputs. (d) ROC curve of Bayesian and conventional neural networks showing area under ROC curves (AUROC).

B. RANDOM NUMBER GENERATOR (RNG)

Considering that the weights in BayesNN follow the Gaussian distribution that $W \sim N(\mu_{i,j}, \sigma_{i,j})$, there are mainly two different realizations of utilizing probabilistic memory arrays as the RNG in prior works.

Directly build the random weight matrix using stochastic memory arrays. In other words, the device conductance follows $N(\mu_{i,j}, \sigma_{i,j})$ directly.

Separate the weights to two deterministic arrays and a probabilistic array following:

$$W_{i,j} = \mu_{i,j} + \sigma_{i,j} \times \delta_{i,j} \quad \text{where} \quad \delta_{i,j} \sim N(0,1) \qquad (1)$$

where $\mu_{i,j}$ and $\sigma_{i,j}$ are mapped to the deterministic array and $\delta_{i,j}$ is generated by the probabilistic array.

The first method could represent the random weights within one group of memory arrays, and the computations are nearly the same as the conventional DNN. However, the arbitrary Gaussian distribution of required weights is a challenge that the average and standard deviation (STD) in hardware depends on the features of memory devices and is hard to control at the same time. An example from [5] used the sum of several RRAM with inherent random noise to achieve the arbitrary Gaussian distribution. The STD of devices is measured at varying average conductance and fitted as a quadratic function. The required mapping state can be determined by solving the equation when the sum of STDs and averages of these devices fit to the targeted Gaussian distribution. Despite the feasibility, the complex preprocessing of solving quadratic equations for each weight element is a huge burden in the actual hardware implementation of large-scale models.

The second method is able to avoid the arbitrary Gaussian by converting the challenge to the standard Gaussian. However, the drawbacks are also obvious that two deterministic arrays to represent $\mu_{i,j}$ and $\sigma_{i,j}$ as well as a probabilistic array to generate $\delta_{i,j}$ are required, causing $\sim 3\times$ memory array overhead. The overhead can be reduced to close to $2\times$ in [6] and [11] by simplifying the matrix of random number δ to a vector of random number ϵ, as shown in Fig. 2(a). However, their conversion makes all the columns in the σ array multiplied with the same vector of random numbers, where the effectiveness of BayesNN cannot be guaranteed without a rigorous mathematical proof.

Based on the related works and their potential problems, we propose to apply the local reparameterization trick [13] to use only a vector of random numbers to implement BayesNN while guaranteeing the uncertainty estimation ability at the same time.

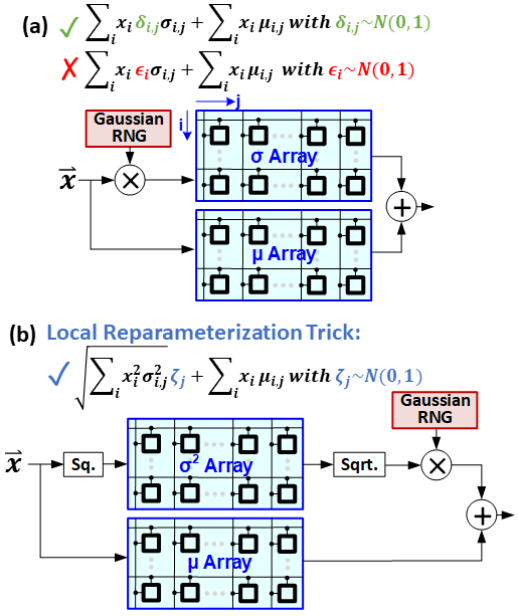

FIGURE 2. (a) Bayesian neural network implementation from [11]. The two crossbar arrays (blue block) behave in the manner of in-memory computing and the Gaussian RNG (red block) is realized by the probabilistic array (p-array). The subscript *i* represents the row dimension and *j* represents the column dimension. The original computation in BayesNN (green equation) is converted to the red one in this implementation, which is inappropriate because directly simplifying the matrix of random numbers δ to a vector ϵ is mathematically unapproved to guarantee the effectiveness of BayesNN on uncertainty estimation. (b) Proposed implementation in this work applying the local reparameterization trick to correctly reduce the dimension of required random numbers. The computation is represented as the blue equation.

III. ALGORITHM-HARDWARE CO-DESIGN OVERVIEW

A. LOCAL REPARAMETERIZATION TRICK

The local reparameterization trick aims to translate the global uncertainty to the local noise to improve the computational and statistical efficiency of BayesNN. The reparameterization leads to much faster convergence, and the sampling cost is highly reduced by sampling activations instead of weights, which is of great importance to energy-efficient hardware. The computation process is shown in Fig. 2(b). The MAC operations $\sum_i x_i \mu_{i,j}$ and $\sum_i x_i^2 \sigma_{i,j}^2$ are separately handled by fixed weight mean array and weight variance array, and then, the Gaussian random numbers from the probabilistic array are multiplied with the outputs from variance array to take the sample and summed with the outputs from the mean array. Recall that the variance of the sum of two independent Gaussian distributions follows $\sigma^2 = \sigma_1^2 + \sigma_2^2$. Therefore, this trick is mathematically equivalent because

$$\begin{aligned} \sum_i x_i N\left(\mu_{i,j}, \sigma_{i,j}\right) &= \sum_i x_i \mu_{i,j} + \sum_i x_i N\left(0, \sigma_{i,j}\right) \\ &= \sum_i x_i \mu_{i,j} + \sqrt{\sum_i x_i^2 \sigma_{i,j}^2}\, N(0,1). \end{aligned} \qquad (2)$$

Based on this trick, only vectors of random numbers ξ with the dimension of the output channel are required. Compared with the random numbers with the dimension of input channel $\times$ output channel to sample each weight, the burden of RNGs can be highly reduced with one dimension canceled. Besides, the hardware cost is further reduced compared with [11] because only one computation of the variance array is required for the first sampling layer, while [11] must compute N times to generate N samples. The mathematical equivalence of the local reparameterization trick ensures no potential impact on the functionality of the algorithm during inference in theory and has been a classical method and widely used in a variety of variational inference algorithms,

but has never been applied to previous BayesNN hardware designs.

B. PROBABILISTIC ARRAY (P-ARRAY)

The RNG in traditional CMOS circuit is usually implemented through linear feedback shift register (LFSR), which costs hundreds of registers and complex logic gates. The intrinsic nonideality and stochasticity of memory devices can be exploited to generate the Gaussian distribution in an energy-efficient manner. In this article, we will use SOT-MRAM as an example of probabilistic bit (p-bit) to build the p-array because of its fast switching speed (sub-ns) and long endurance ($>5 \times 10^{10}$) as demonstrated in industry-grade 300-mm wafers [14].

MTJ consists of two layers of magnetic metal separated by a spacer, where the magnetization of one of the layers is pinned in a particular direction, while the direction of the other layer can be manipulated by a spin current or an external magnetic field. The device exhibits high resistance when the magnetizations have the opposite direction (antiparallel) and presents low resistance on the contrary (parallel). SOT-MRAM switches the free magnetic layer by injecting an in-plane current in an adjacent SOT metal layer so that the write path is separated from the read path. Garello *et al.* [14] and [15] and Doevenspeck *et al.* [16] and [17] showed that the switching probability of SOT-MRAM can be controlled by the applied programming pulsewidth and the programming voltage. There are also follow-up works [4], [18], [19] to intentionally lower the nanomagnet barrier to create a more thermally instable p-bit such as replacing the usual elliptical magnets with circular ones.

In this probabilistic array, the Gaussian random numbers can be generated following a similar scheme as the traditional CIM except that all the read select (RS) lines should be turned on irrelevant to the input data. The column currents are sensed by an analog-to-digital converter (ADC) from the source line (SL), naturally generating samples from binomial distribution when the state of each device follows the Bernoulli distribution depending on the spin probability. The binomial can be approximated as Gaussian distribution when the number of variables is large enough. Therefore, a vector of approximated Gaussian random numbers can be generated during a parallel readout of the probabilistic array with a sufficient number of rows.

The typical SOT-MRAM cell in crossbar array usually employs two access transistors besides the MTJ device for read and write control. However, in our RNG design, all the cells can be programmed at the same time given the same voltage and can be read out with all rows turned on to directly sum up the current along the columns. Therefore, the two access transistors can be eliminated in this fully parallel read and write mode, with the read/write signals exactly the same for each cell, as shown in Fig. 3(a). The peripheral circuits of p-array are also simplified compared with general CIM array as shown in Fig. 3(b), where the row/column decoder can be omitted because of the fully parallel operation.

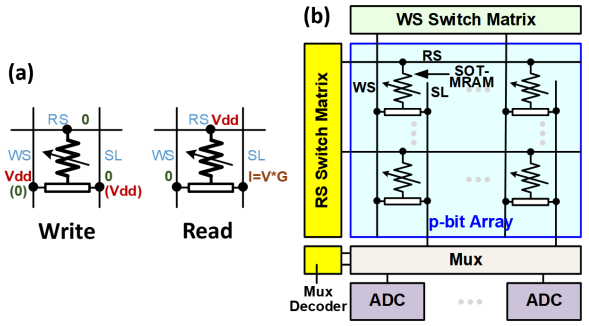

FIGURE 3. Schematic of p-bit and p-array. (a) Voltage bias schemes in the write and read operation of SOT-MRAM. (b) SOT-MRAM-based p-bit crossbar array in fully parallel read/write mode with minimal peripheral circuits.

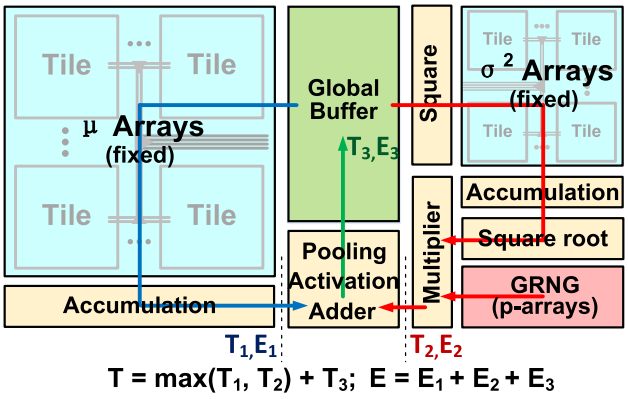

FIGURE 4. Overall architecture of the proposed Bayesian neural network accelerator. The computation inside mean arrays and variance arrays can be operated in parallel.

The overall architecture of our proposed BayesNN hardware is shown in Fig. 4. The computation inside mean arrays and variance arrays can be operated in parallel, and thus, the overall latency is the maximum of them two summed up with the following activation, pooling, and so on. The square and square root modules are implemented based on lookup table for quick computation. The interconnects between memory arrays are based on H-tree routing. The time and energy costs of data transmission and the peripheral digital modules have been included in this evaluation and could be referred to [20]. The basic operations of CIM scheme and its hardware support could be further referred to [2].

IV. SAMPLING STRATEGY

A challenge of BayesNN is the extensive operations caused by multiple sampling trials. To reduce the required operations as much as possible, we first measured the effects of number of samples on accuracy, uncertainty calibration, OOD detection, and hardware cost. As shown in Fig. 5, for in-distribution figures, the accuracy and ECE both get improved with more samples, but they tend to saturate after 10–20 samples. The figures rotated with 120° are used as the example for the

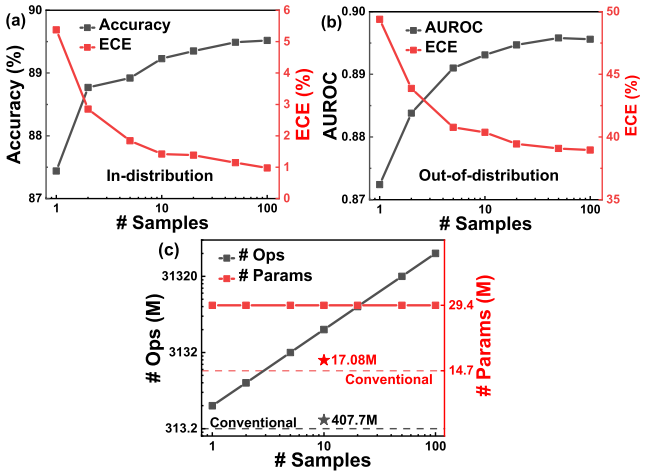

FIGURE 5. Effects of number of samples. (a) Accuracy and ECE of regular ID inputs versus number of samples. (b) AUROC and ECE of OOD inputs (120° rotated CIFAR-10 images) versus number of samples. (c) Number of operations and parameters versus number of samples. Those of conventional DNN are labeled as a dashed line. The stars represent those of BayesNN with sampling strategy after being optimized in Section IV.

OOD tests and a similar trend happens on AUROC and ECE. As for hardware cost, the number of operations grows linearly with the number of samples, which means that the latency and energy consumption would increase at the same rate. The number of parameters is twice of conventional DNN to store both weight arrays and variance arrays, which primarily introduces the overhead of chip area.

Sampling only a few layers can be an effective way to reduce the enormous number of operations of BayesNN. Some possible strategies are shown in Fig. 6(a). Sampling ten times is utilized in the following discussion. The black and red lines, respectively, represent no sampling and sampling for specific layers, and the number of operations of each layer compared with the conventional DNN is labeled as $\times N$.

One choice is sampling the first few layers. Fig. 6(b) shows the ECE of in-distribution and OOD inputs with respect to the number of sample layers. The calibration errors quickly get reduced with an increasing number of sampling layers when only the first few layers get sampled. However, the number of operations is still tremendous because there are still ten tracks to compute separately for later layers even without sampling. The other choice is to sample the last few layers, where potentially the number of operations can be highly reduced; however, as shown in Fig. 6(b), more than half of the layers requires to be sampled to get the calibration errors effectively reduced for both in-distribution and OOD figures, which in fact achieves trivial operation reduction. In other words, sampling the last few layers is favorable to reduce the operations, but the diversity of output is not as good as sampling the first few layers to ensure the uncertainty calibration.

In order to take advantage of the benefits of both strategies, we propose to sample the last layers with large variance to

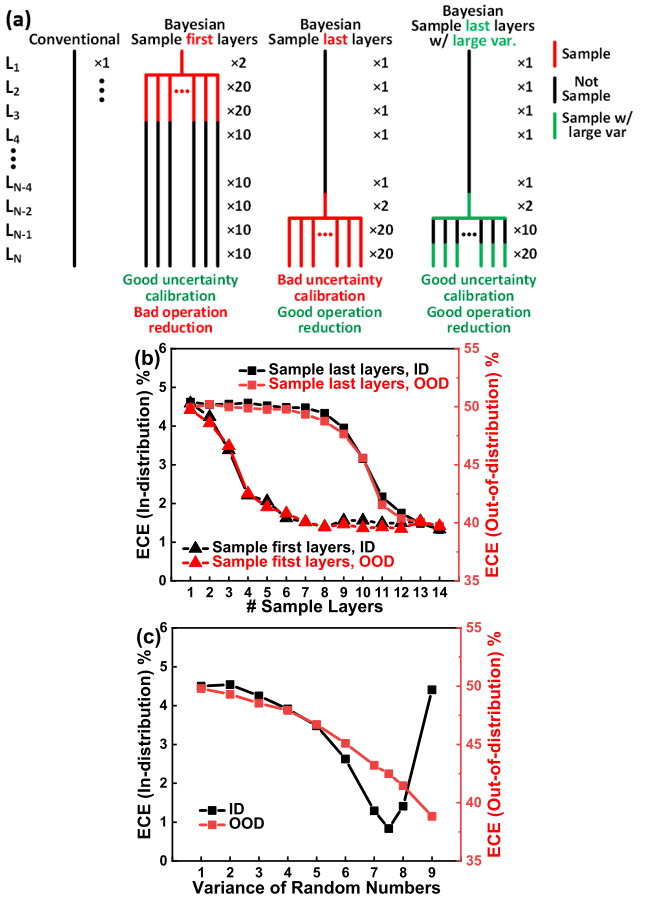

FIGURE 6. (a) Computation flow of conventional DNN and different sampling strategies of BayesNN. (b) ECE of ID and OOD inputs versus number of sample layers with only first few layers or last few layers sampled. (c) ECE of ID and OOD inputs versus variance of Gaussian random numbers with only last layers sampled.

promote the diversity, and thus, the good uncertainty calibration and good operation reduction can be achieved at the same time. In other words, the operation is changed to

$$\sum_i x_i \mu_{i,j} + \sqrt{\sum_i x_i^2 \sigma_{i,j}^2}\, N(0,k) \tag{3}$$

where k is the increased rate of the variance. The Gaussian random numbers with larger variance $N(0, k)$ can be simply implemented based on the standard Gaussian by multiplying the variance value. For instance, the last and 3rd-to-the last layers are sampled with varying Gaussian distribution variance as shown in Fig. 6(c), where the variance (k) is increased by 9×. The ECE of ID inputs gradually gets reduced with increased variance, but it suddenly becomes worse after a threshold (variance = 7.5 in this example) because the deviations become too large to keep a basic classification ability, while the ECE of OOD figures is able to keep reduced with larger variance. Considering the tradeoffs, we will take the threshold as the optimal variance to adopt. With this strategy, the uncertainty calibration can be maintained, while the

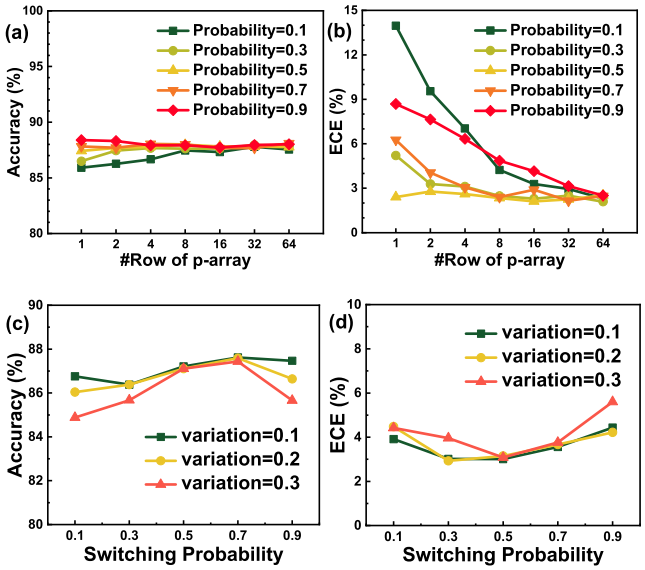

FIGURE 7. (a) Accuracy and (b) ECE versus number row of the probabilistic array (RNG) and switching probability of "1." (c) Accuracy and (d) ECE versus switching probability of "1" and the cell-to-cell variation of the probability.

number of operations and parameters could approach to them of conventional DNN, as labeled in Fig. 5(c). However, we have to declare that the accuracy is slightly dropped to 88.31% as a penalty of this sampling strategy.

V. PERFORMANCE BENCHMARKING

We develop an end-to-end simulator framework based on DNN + NeuroSim [21] for fast early-stage design space exploration of CIM and p-array-based BayesNN accelerators, supporting flexible and hierarchical design options from device level (transistor and memory properties) to circuit level (array architecture with peripheral circuit modules) and up to algorithm level (neural network topologies). It can predict the hardware performance metrics, such as area, latency, dynamic energy and leakage power consumption, and algorithm inference accuracy and uncertainty estimation with hardware nonideal effects.

A. NONIDEALITIES OF P-ARRAY

The switching probability of MTJ may have a built-in bias that deviates from 50% under the applied programming voltage and pulsewidth, which might affect the approximation of generated random numbers to true Gaussian random numbers. Fig. 7(a) and (b) shows the accuracy and ECE of in-distribution inputs measured with a varying number of rows of p-array and the expected switching probability of "1." The accuracy tends to saturate around 88% when the p-array has more than eight rows. For ECE, the calibration error keeps low with balanced 50% switching probability, while with biased probability, large errors are introduced with a few numbers of rows. With the increasing number of rows of the p-array, the ECE gradually becomes almost the same regardless of the switching probability. The results are reasonable because the binomial distribution $B(n,p)$ is more approximated to the Gaussian distribution when n increases and p approaches 50%. To clarify, the variance of the random numbers [k in (3)] needs to be tuned for different cases, and it is 10.5 in the above simulation. A guideline to help find the threshold variance is one of the future works.

In addition to the possible bias, there may also be some cell-to-cell variations of the switching probability caused by process variation. We keep the p-array as eight rows and measure the performance with varying switching probability of "1" and its variation, as shown in Fig. 7(c) and (d). With a well-balanced (50%) switching probability, both the accuracy and ECE are able to keep a good performance, but the variation would affect the results more severely with a more biased switching probability. To clarify, there are other unintentional variations (such as the conductance variation across the cells) that are not included in this simulation but could also degrade the performance and should be evaluated in future work.

B. HARDWARE PERFORMANCE BENCHMARK

We evaluate the hardware performance of our proposed design with benchmark settings listed in Table 1. There are totally 179-Mb deterministic arrays to support the setting, with 158 Mb for the mean weights (μ_{ij}) and only 21 Mb for the variance weights (σ_{ij}^2) as only those of the last layers are in use.

TABLE 1. Benchmark settings.

22nm SOT-MRAM [16]			Circuit & System		
Cell size	p-array	6F×6F	Array size & ADC precision	p-array	8×128 & 3-bit
	general array	8F×20F		general array	128×128 & 8-bit SAR-ADC
Ron/Roff		6M/15M	Input precision		8-bit
Write current		980uA	Weight precision		8-bit
Write pulse width		1ns	Model		VGG16
Write voltage		0.5V	Dataset		CIFAR-10

The probabilistic arrays generate the Gaussian random numbers with throughput matched with the deterministic array, requesting capacity of only 12.3 Kb. The hardware overhead of BayesNN is compressed as much as possible in this design. The detailed breakdown of array sizes is listed in Table 2. The device parameters of the 22-nm industry-grade SOT-MRAM [16] are utilized in both p-array and general crossbar array, but the cell size is smaller in p-array because of the elimination of access transistors. In our simulation, an individual p-array with peripheral circuits costs area of 0.0013 mm^2 (8 rows × 128 columns), read dynamic energy of 2.11 pJ, and write dynamic energy of 502.12 pJ. It should be pointed out that this SOT-MRAM device is specially engineered for CIM application with high on-state resistance (∼6 MΩ), which contributes to an ultralow read dynamic energy. Hence, the programming energy is the main dominant factor. However, on system level, the differences of

TABLE 2. Layer-by-layer array size breakdown.

Deterministic array for mean weights (μ_{ij})			
Layer	Array Size (Mb)	Layer	Array Size (Mb)
1	2.36[1]	8	18.87
2	2.36	9	18.87
3	4.72	10	18.87
4	4.72	11	18.87
5	9.44	12	18.87
6	9.44	13	18.87
7	9.44	14	2.36
Deterministic array for variance weights (σ_{ij}^2)			
Layer	Array Size (Mb)	Layer	Array Size (Mb)
12	18.87	14	2.36
Probabilistic array for random numbers generation: 12.3Kb			

[1]2.36Mb is the minimum array size for one layer based on our assumption of memory array hierarchy.

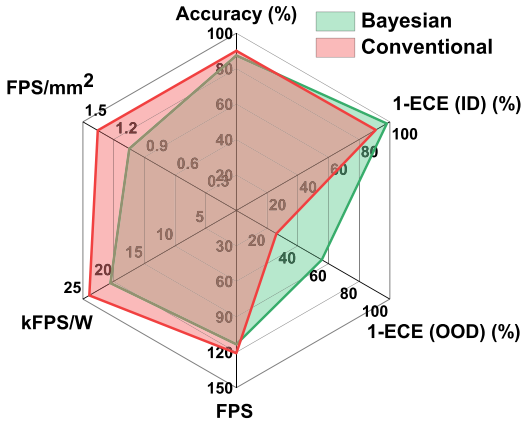

FIGURE 8. Comparison of accuracy, ECE of ID and OOD inputs, throughput (FPS), energy efficiency (kFPS/W), and compute density (FPS/mm^2) between BayesNN and conventional DNN.

SOT-MRAM devices are in fact invisible benefitting from the tremendous sampling reduction achieved in the co-design discussed in Sections III and IV.

The overall comparison between Bayesian and conventional DNN is shown in Fig. 8. The accuracy of BayesNN is comparable to the conventional one and has much better uncertainty calibration for both in-distribution and OOD inputs. With our proposed hardware-friendly design, the throughput, energy efficiency, and compute density of BayesNN are slightly undermined compared to the conventional counterpart. This is due to a few more operations and increased chip area to hold the variance arrays and a few additional logic modules. The hardware penalties are inevitable as the more intensive computation of the algorithm in exchange for the uncertainty. The tradeoff is valuable in some scenarios such as autonomous driving, where an overconfident prediction can be very harmful, and thus, detecting a high uncertainty and giving the control back to human are a better choice.

TABLE 3. Performance comparison table.

Param	[8] (digital)	[9] (digital)	[11]	This work
Model & Dataset	Two hidden layers with 200 neurons (784-200-200-10) on MNIST			VGG16 on Cifar-10
#Samples	10[1]	500[2]	10	10[3]
#Ops (M)	1.99	6.57	1.99	407.67
Throughput (TOPS)	0.639	0.068	------	0.092
Energy efficiency (TOPS/W)	0.105	0.143	2.516	10.387

[1]The number of samples in [8] is not declared, so we assume it as 10 samples to derive the throughput and energy efficiency.
[2]Ref. [9] takes separately 10, 10 and 5 samples for the three layers and 500 samples in total.
[3]Our work takes 10 samples only on the last and the 3rd-to-the last layers with large variation.

Table 3 shows a comparison with state-of-the-art BayesNN hardware design in the literature. It should be pointed out that previous works were all tested on the MNIST dataset, while our work is the first one to extend the design toward more complex CIFAR-10 dataset. For a fair comparison across datasets, the metrics are converted to TOPS and TOPS/W. Our work achieves 73× and 4× energy efficiency advantages compared with previous digital design [9] and p-array-based design [11].

VI. CONCLUSION

In this work, a software–hardware co-design and a device-circuit-algorithm full-stack benchmark framework for BayesNN accelerators are developed. The local reparameterization trick is proposed to be applied for the first time on BayesNN hardware accelerators to substantially reduce the burden of RNG, with the MAC operations processed by general crossbar arrays and only vectors of Gaussian random numbers generated by probabilistic arrays. The sample strategies with about ten samples and only sample the last few layers with larger variation is optimal considering the tradeoffs of accuracy, uncertainty calibration, and hardware efficiency. The effects of device nonidealities such as the bias and variation of switching probability are simulated, concluding that 50% probability is the best with trivial effects of number of rows of p-array or variation of the switching probability, while for devices with more biased probability, the greater number of rows and less variation is required to achieve a good performance. The hardware performance in array level and system level are also performed. Compared with conventional DNN, BayesNN could achieve comparable accuracy with slight penalty on throughput, energy efficiency, and compute density, but much better uncertainty calibration. Compared with previous BayesNN hardware designs, our work successfully improves the energy efficiency from 4× to 100×.

REFERENCES

[1] H. Wang and D.-Y. Yeung, "Towards Bayesian deep learning: A framework and some existing methods," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 12, pp. 3395–3408, Dec. 2016.

[2] S. Yu, H. Jiang, S. Huang, X. Peng, and A. Lu, "Compute-in-memory chips for deep learning: Recent trends and prospects," *IEEE Circuits Syst. Mag.*, vol. 21, no. 3, pp. 31–56, 3rd Quart., 2021.

[3] M. Yamaoka *et al.*, "A 20k-spin Ising chip to solve combinatorial optimization problems with CMOS annealing," *IEEE J. Solid-State Circuits*, vol. 51, no. 1, pp. 303–309, Dec. 2015.

[4] K. Y. Camsari *et al.*, "Stochastic p-bits for invertible logic," *Phys. Rev. X*, vol. 7, no. 3, 2017, Art. no. 031014.

[5] Y. Lin *et al.*, "Bayesian neural network realization by exploiting inherent stochastic characteristics of analog RRAM," in *IEDM Tech. Dig.*, Dec. 2019, p. 14.

[6] A. Malhotra, S. Lu, K. Yang, and A. Sengupta, "Exploiting oxide based resistive RAM variability for Bayesian neural network hardware design," *IEEE Trans. Nanotechnol.*, vol. 19, pp. 328–331, 2020.

[7] H. Mulaosmanovic *et al.*, "Random number generation based on ferroelectric switching," *IEEE Electron Device Lett.*, vol. 39, no. 1, pp. 135–138, Nov. 2017.

[8] R. Cai *et al.*, "VIBNN: Hardware acceleration of Bayesian neural networks," in *Proc. 23rd Int. Conf. Architectural Support Program. Lang. Erating Syst. (ASPLOS)*. New York, NY, USA: ACM, 2018, pp. 476–488.

[9] X. Jia *et al.*, "Efficient computation reduction in Bayesian neural networks through feature decomposition and memorization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 4, pp. 1703–1712, May 2020.

[10] H. Fan, M. Ferianc, M. Rodrigues, H. Zhou, X. Niu, and W. Luk, "High-performance FPGA-based accelerator for Bayesian neural networks," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 1063–1068.

[11] K. Yang, A. Malhotra, S. Lu, and A. Sengupta, "All-spin Bayesian neural networks," *IEEE Trans. Electron Devices*, vol. 67, no. 3, pp. 1340–1347, Mar. 2020.

[12] C. Blundell *et al.*, "Weight uncertainty in neural network," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1613–1622.

[13] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 2575–2583.

[14] K. Garello *et al.*, "SOT-MRAM 300 mm integration for low power and ultrafast embedded memories," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 81–82.

[15] K. Garello *et al.*, "Manufacturable 300 mm platform solution for field-free switching SOT-MRAM," in *Proc. Symp. VLSI Technol.*, Jun. 2019, pp. 194–195.

[16] J. Doevenspeck *et al.*, "SOT-MRAM based analog in-memory computing for DNN inference," in *Proc. IEEE Symp. VLSI Technol.*, Jun. 2020, pp. 1–2.

[17] J. Doevenspeck *et al.*, "Multi-pillar SOT-MRAM for accurate analog in-memory DNN inference," in *Proc. IEEE Symp. VLSI Technol.*, Jun. 2021, pp. 1–2.

[18] P. Debashis, R. Faria, K. Y. Camsari, and Z. Chen, "Design of stochastic nanomagnets for probabilistic spin logic," *IEEE Magn. Lett.*, vol. 9, pp. 1–5, 2018.

[19] V. Ostwal and J. Appenzeller, "Spin–orbit torque-controlled magnetic tunnel junction with low thermal stability for tunable random number generation," *IEEE Magn. Lett.*, vol. 10, pp. 1–5, 2019.

[20] A. Lu, X. Peng, Y. Luo, and S. Yu, "Benchmark of the compute-in-memory-based DNN accelerator with area constraint," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 9, pp. 1945–1952, Sep. 2020.

[21] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu, "DNN+NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," in *IEDM Tech. Dig.*, Dec. 2019, p. 32.