SpinBayes: Algorithm-Hardware Co-Design for Uncertainty Estimation Using Bayesian In-Memory Approximation on Spintronic-Based Architectures

Recent development in neural networks (NNs) has led to their widespread use in critical and automated decision-making systems, where uncertainty estimation is essential for trustworthiness. Although, conventional NNs can solve many problems accurately, does not capture the uncertainty of data or the model during optimization. In contrast, Bayesian neural networks (BNNs), which learn probabilistic distributions for their parameters, offer a sound theoretical framework for estimating uncertainty. However, traditional hardware implementations of BNNs are expensive in terms of computational and memory resources, as they (1) are realized with inefficient von Neumann architectures, (2) uses a significantly large number of random number generators (RNGs) to implement the distributions of BNN, and (3) has a substantially greater number of parameters than conventional NNs. The post-von-Neumann computing-in-memory (CiM) architectures with spintronics memories are typically used to accelerate classical NNs due to their inherent parallelism and low power consumption of spintronics devices. Therefore, implementing BNNs into CiM architectures is highly attractive for efficient implementation. However, implementing the learned probabilistic distributions of BNN to CiM hardware directly may not be feasible or can incur high overhead. In this work, we propose an *in-memory approximation* that allows efficient implementation of Bayesian distribution to CiM hardware, and a Bayesian network called *SpinBayes* for efficient sampling during Bayesian inference. Compared to state-of-the-art methods, the memory overhead is reduced by 8× and the energy consumption by 80×. Additionally, our method without sacrificing performance (as evaluated on several classification and segmentation tasks) can detect up to 100% of various types of out-of-distribution data, highlighting the robustness of our approach.

CCS Concepts: • **Hardware → Spintronics and magnetic technologies**; • **Computing methodologies → Artificial intelligence**.

Additional Key Words and Phrases: Bayesian neural network, Spintronic, Uncertainty estimation, low cost uncertainty estimation

ACM Reference Format:

. 2018. SpinBayes: Algorithm-Hardware Co-Design for Uncertainty Estimation Using Bayesian In-Memory Approximation on Spintronic-Based Architectures. In . ACM, New York, NY, USA, 17 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Recent advances in artificial intelligence (AI) methods, and the availability of huge amounts of data, have led to excellent performance in various domains such as computer vision, speech recognition, and natural language processing. Subsequently, AI methods such as Neural Networks (NNs) are widely utilized for all types of inference and decision-making. A NN attempts to identify underlying relationships in a set of data by mimicking the way the human brain works to make predictions [14]. However, these predictions may be affected by noise and model prediction errors, leading to uncertainty in prediction. Therefore, it is crucial to evaluate the reliability and efficiency of AI systems before deploying them in real-world scenarios. Hence, quantifying the uncertainty of a prediction is highly desirable.

In NNs, uncertainty estimation is the process of quantifying the confidence in a network's prediction. Such estimation is vital for trustworthiness, since it gives a measure of reliability and resilience for decision-making [10]. Uncertainty quantification is essential for practical applications, especially when NNs make automated decisions in domains like

Manuscript submitted to ACM

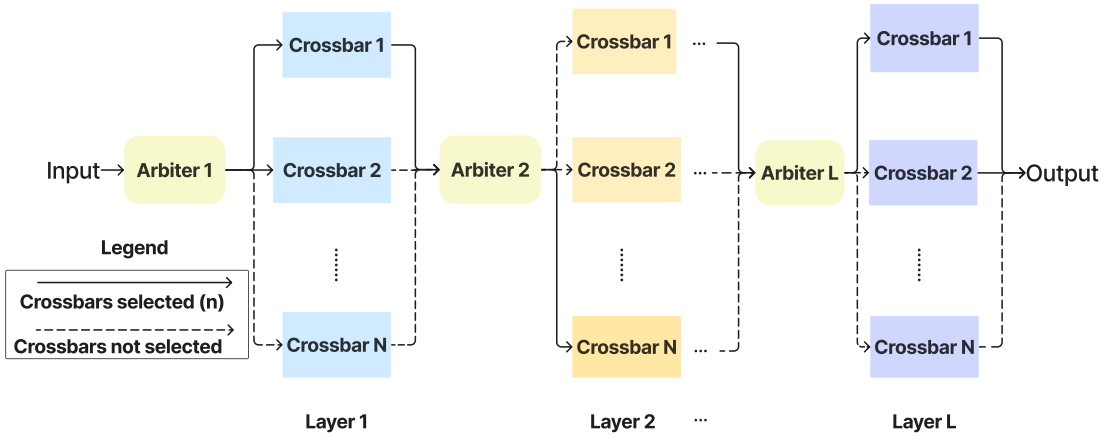


Fig. 1. Network topology of the proposed SpinBayes.

robotic control, robotic-assisted surgery, and autonomous driving. Otherwise, uninformed prediction can be detrimental [3].

Conventional NNs optimize a set of single-point parameters that map input data to output predictions or classifications. Therefore, cannot measure predictive uncertainty and often generate overconfident predictions even when they are incorrect or uncertain about the prediction [1].

On the contrary, probabilistic *Bayesian Neural Networks* (BNNs) represent the parameters as distributions instead of deterministic values [21]. Thus, BNNs induce a predictive distribution over the output values, which inherently provides uncertainty estimates. Thus, BNN provides robust and reliable predictions. Yet, conventional hardware implementations of BNNs pose significant computational and memory resource challenges. Firstly, BNNs are implemented on inherently inefficient computation-centric von Neumann architectures, which, due to the memory bottleneck, consume a considerable amount of computing power and energy [22]. Second, they require numerous random number generators (RNGs) to accurately model the parameter distributions of BNN, which further increases resource requirements. The intrinsic complexity of BNNs necessitates a significantly larger number of parameters than traditional neural networks. As a result, BNNs are impractical for numerous applications with low computational costs and high-performance requirements.

To accelerate NN workload execution, dedicated hardware accelerators such as Tensor Processing Units (TPUs), Field Programmable Gate Arrays (FPGAs), and *Compute-in-Memory* (CiM) have been proposed [23]. Among them, CiM architectures are particularly appealing as they can perform computations within the memory array itself, which eliminates data transfer [2]. CiM architecture utilizes crossbar structures with thousands of memristive synaptic devices to implement the NN functionality [24]. Among the memristive devices, the Magnetic Random-access Memory (MRAM) technology is very appealing as it offers a high writing speed (few *ns*), quasi-infinite endurance, and low power consumption [24]. Therefore, implementing BNNs to CiM architectures can lead to a highly efficient solution. Unfortunately, implementing learned distributions of BNN to CiM architectures is challenging due to their deterministic and parallel computing nature. Additionally, fly sampling for BNN inference is difficult to implement efficiently that requires sufficiently small number of RNGs.

In this paper, we co-design the algorithm and hardware to solve some of the challenges of implementing BNNs to CiM architectures for efficient uncertainty estimation and reduce the inherent overhead of BNN. The overall contributions are:

- We propose a *Bayesian in-memory approximation* approach that permits the implementation of learned distributions of BNN to CIM based hardware accelerator with spintronic devices,
- Also, a stochastic topology called *SpinBayes* is proposed that enables efficient on-the-fly sampling for Bayesian inference with a significantly smaller number of RNGs, During each forward pass, different weights are drawn from the mapped posterior distribution.
- The combined effect of our approach allows us to offset some of the inherent costs of BNNs. Compared to existing BNN implementation, our spintronic-based implementation requires only 0.26 µJ/Image for Bayesian inference, which is up to 800× lower.
- Additionally, to reduce the inherent memory overhead of the proposed SpinBayes topology and to represent the parameters of our proposed *SpinBayes Network* in spintronic devices, we quantize the parameters of our BNN topology. Consequently, our method requires 8× lower memory overhead compared to existing BNN algorithms.
- In conjunction, we propose a multi-level crossbar array structure based on the spintronic devices to represent quantized weights. The stochastic component of Bayesian inference is realized by harnessing the inherent stochasticity of Spin-Orbit Torque devices.
- In terms of uncertainty estimates, which is one of the main motivations of BNN, our proposed approach can attain similar performance on several uncertainty estimation metrics and can detect up to 100% out-of-distribution data.
- The proposed method is thoroughly evaluated for its effectiveness on a number of difficult semantic segmentation and classification datasets and large NN topologies.

The organization of the paper is as follows: Section 2 covers the basic background of our work. Section 3 introduces our proposed SpinBayes and provides details on their hardware implementation. Then, in Section 4 we give a detailed multi-level evaluation of our proposed method, and in Section 5, we wrap up the paper.

2 BACKGROUND

2.1 Spintronic Technology

Magnetic Tunnel Junction (MTJ) is the primary memory component of any magnetic memory (MRAM). It consists of two ferromagnetic layers: the Free Layer (FL) and the Reference Layer (RL). Both layers are separated by a thin insulating layer. The relative magnetization of each layer, e.g., parallel (P) or anti-parallel (AP), determines the overall resistance state of the device. The primary physical mechanisms underlying the magnetization switching are spin transfer torque (STT) and Spin-Orbit Torque (SOT), also known as STT-MRAM and SOT-MRAM, respectively. STT-MRAM devices offer many advantages in terms of write speed (a few ns), endurance, and low power consumption [7]. However, such technology is not convenient to perform Matrix-Vector Multiplication operations in the conventional crossbar array due to its low resistance levels (i.e., few $k\Omega$). In fact, for analog computation, all bitcells are read at the same time in a traditional crossbar design. Therefore, the low resistive STT memory creates an uncontrollable current at the crossbar's output. To address this issue, one can take advantage of the SOT-MRAM devices, which offer a resistance that can be tuned to up to dozens of MΩ resistance [24]. SOT-MRAM is a three-terminals device that consists of an MTJ mounted on a heavy metal substrate. Lately, it was proposed to implement multiple MTJs on the same heavy metal layer to

emulate a multi-value cell [8]. Such a device combines the Voltage Control Magnetic Anisotropy (VCMA) effect in conjecture with the SOT and allows writing multi-bit values inside a single cell.

2.2 Bayesian Neural Networks (BNNs)

A neural network can be seen as a parametric function $\mathbf{f} : \mathbb{R}^D \times \mathbb{R}^P \rightarrow \mathbb{R}^C$, where D, P and C denote the dimensions of the inputs $\mathbf{x}$, learnable parameters $\boldsymbol{\theta}$ and outputs $\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})$ respectively. The task of learning an NN relates to adjusting the parameters $\boldsymbol{\theta}$, such that $\mathbf{f}(\mathbf{x}_m, \boldsymbol{\theta}) \approx \mathbf{y}_m$ for input-output pairs for a given training dataset $\mathcal{D} = \{(\mathbf{x}_m, \mathbf{y}_m) \mid m = 1, \cdots, M\}$ $\mathcal{N}(\mathbf{y}; \mathbf{f}(\mathbf{x}, \boldsymbol{\theta}), \mathbf{I} \cdot \tau)$ through maximum likelihood estimation. Specifically, given a chosen likelihood $p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})$, for example, $p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}) = \text{Categorical}(\mathbf{f}(\mathbf{x}, \boldsymbol{\theta}))$ in case of classification, the likelihood given the data $\mathcal{D}$ is maximized. The resulting $\boldsymbol{\theta}$ can then be seen as a point estimate, and can be used to obtain point estimates of $\mathbf{y}$ through the likelihood.

In contrast to this, Bayesian NNs consider a distribution of possible parameter vectors $\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$. Here, learning relates to estimating the posterior distribution $\boldsymbol{\theta} \sim p(\boldsymbol{\theta} \mid \mathcal{D})$ given some data $\mathcal{D}$. This allows to obtain a posterior distribution $\mathbf{y}^*$ over test points $\mathbf{x}^*$ as

$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) = \int p(\mathbf{y}^* \mid \mathbf{x}^*, \boldsymbol{\theta})\, p(\boldsymbol{\theta} \mid \mathcal{D})\, \mathrm{d}\boldsymbol{\theta}. \tag{1}$$

Through the posterior distribution, we can not only obtain point estimates but also assess uncertainties regarding our predictions. Unfortunately, training Bayesian NNs is not as straightforward as training classical NNs. Specifically, $p(\boldsymbol{\theta} \mid \mathcal{D})$ can usually not be obtained in closed form, and approximation techniques, like variational inference, need to be employed. In variational inference, a computationally convenient (variational) distribution $q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \approx p(\boldsymbol{\theta} \mid \mathcal{D})$ with (variational) parameters $\boldsymbol{\omega}$ is used in place of $p(\boldsymbol{\theta} \mid \mathcal{D})$. Common choices for $q_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ are Gaussians with diagonal covariance matrices. In this case, $\boldsymbol{\omega}$ summarizes the mean $\boldsymbol{\mu}_\omega$ and variances $\boldsymbol{\sigma}^2_\omega$. The variational parameters of $q_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ are found by minimizing the Kullback-Leibler divergence $\text{KL}\left(q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \,\|\, p(\boldsymbol{\theta} \mid \mathcal{D})\right)$ with respect to $\boldsymbol{\omega}$, see [5].

After $q_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ is acquired this way, predictions of $\mathbf{y}^*$ for given $\mathbf{x}^*$ can be obtained by Monte Carlo approximation using T samples, as

$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) \approx \frac{1}{T} \sum_{t=1}^{T} p\left(\mathbf{y}^* \mid \mathbf{x}^*, \boldsymbol{\theta}^{(t)}\right) \quad \text{with} \quad \boldsymbol{\theta}^{(t)} \sim q_{\boldsymbol{\omega}}(\boldsymbol{\theta} \mid \mathcal{D}). \tag{2}$$

In this work, as our goal is the hardware implementation of the above-mentioned paradigm, we propose a more *CiM accelerator-friendly approximation* of posterior inference for $p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D})$.

2.3 Uncertainty Estimation in Deep Learning

Uncertainty estimation in deep learning aims to measure and quantify the reliability of deep learning models in practical applications. Uncertainty estimation can help identify errors, outliers, adversarial attacks, and data distribution shifts. There are different types of uncertainties that can affect the predictions of deep learning models. Aleatoric uncertainty arises due to inherent randomness or noise in the data. On the other hand, epistemic uncertainty appears due to a mismatch between training and test data (also called out-of-distribution data or OoD) or lack of knowledge or data. Uncertainty can also be due to conflicting or insufficient evidence, (known as evidential uncertainty).

How a deep learning model behaves in an uncertain situation is important for a number of applications. For example, if the NN receives an OoD, a trustworthy system should predict it with high uncertainty. Thus, uncertainty estimates information in the prediction can be incorporated into the decision-making procedure.

2.4 Related Works

2.4.1 Algorithmic Approaches. Multiple approaches for Bayesian uncertainty estimation of NNs exist in the literature. For example, MC-Dropout [9], and MC-DropConncet [20] are some of the most popular methods for estimating uncertainty for a NN trained with either neuron or weights dropout layer and weight decay regularization. However, not all NN topologies employ dropout as a regularization method, and hardware implementation of dropout can be challenging. Since the batch normalization method is more commonly used instead of dropout, [25] proposed MC-Batchnorm. Their approach requires passing a randomly sampled mini-batch from the training data through the NN and re-calculating batch statistics. However, as a drawback, this method requires storing the training datasets in the hardware. Additionally, processing each mini-batch requires a great number of matrix-vector multiplication operations, which is costly.

Alternatively, ensemble learning techniques are also popular. For uncertainty estimation, ensemble approaches combine the predictions of multiple distinct deterministic networks. For example, the Deep ensemble method [15] trains several networks with different random initializations. Consequently, it has high memory consumption during training, as several models need to be trained. Also, the ensemble method provides deterministic uncertainty and predictions. Therefore, the benefits of BNNs, e.g., generating unlimited ensembles from a single-trained model, and post-training model compression, might not be possible. The Variational inference-based approach such as [5] introduced backpropagation-compatible Bayes by Backpropagation (BB-BackProp) algorithm. BB-BackProp algorithm allows for learning a probability distribution for the weights of a NN. Although, the algorithm is efficient, and principled, implementing its probability distribution and sampling from it during forward propagating in CiM hardware is challenging. In this paper, we attempt to solve this problem.

2.4.2 Hardware Implementations of BNNs. Typically, both Bayesian and ensemble methods, are implemented in conventional architectures. Some studies propose implementing BNNs on hardware accelerators. Work [4], proposed an FPGA implementation with quadratic nonlinear activation functions. Even though the uncertainty estimate is obtained with a single forward pass on a small NN, their method may not be applicable to larger networks or datasets. In [18], the authors proposed a CiM implementation where the crossbar array stores the variance parameter. In addition, several stochastic resistive (RRAM) devices were used to get the probability distribution sampling. However, their method requires one stochastic unit per input, which is energy inefficient. Work in [27] suggested the use of three crossbar arrays for BNN implementation with low-barrier MTJs. The first array is used for the sampling operation, and the other two represent the required mean and variance for BNN inference. Although they show reduced energy consumption, the low energy barrier of the memories limits their endurance, which can lead to a decrease in accuracy in the CiM engine.

Our attempt to use SOT-MRAM-based implementation exploits the memory-like behavior of the MTJ as a deterministic weight, and its inherent stochasticity to implement the stochastic component required for the probabilistic sampling. Our method is significantly better in terms of memory overhead and energy consumption compared to the aforementioned studies, as presented further in Section 4.3.

3 METHODOLOGY

3.1 Limitation of BNNs for CiM Implementation

As training can be done in software, the main challenge for BNNs is the realization of posterior distribution on hardware and sampling from it for the inference step given $\mathbf{y}^*$, see equation (2). Specifically, samples $\boldsymbol{\theta} \sim q_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ need to be drawn in each forward pass of Bayesian inference. Multiply-and-accumulate operations for the inference are done on the samples. Unfortunately, such on the fly sampling can lead to high run-time computational costs due to the nature of implementation.

A general tendency is realizing the $q_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ (assuming variational distributions) with Gaussian random number generators (GRNGs). However, such implementation will require a GRNG for each parameter (weights), which can be millions. Similarly, for MC-Dropconnect-based approximation [20], Bernoulli random number generators (BRNGs) are required for each parameter. For MC-Dropout-based approximation [9] and variational distribution at the neuron dimension, a GRNG or BRNG is required for each neuron. Such implementations are again not scalable, as the number of neurons can be in the millions as well. Furthermore, due to the deterministic nature and limited bit precision of spintronic memories on CiM-based hardware accelerators, only discrete values can be stored with a fixed number of bits. Bayesian sampling and parameter representation can not be realized directly in an efficient manner.

In this paper, we propose a memory-centric Bayesian approximation approach called *in-memory approximation* that allows direct mapping of the posterior distribution to crossbar arrays of CiM architectures. Also, to sample from the CiM architecture, we propose a novel BNN topology, *SpinBayes*.

3.2 Bayesian In-Memory Approximation

Our proposed *in-memory approximation* method approximates $q_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ such that it can efficiently be mapped to and sampled by a CiM-hardware implementing BNNs. Specifically, we take N samples from the high precision with continuous values range $q_{\boldsymbol{\omega}}(\theta)$ and map them to N parallel crossbars for a layer. Therefore, N parallel crossbars represent each layer.

However, spintronic devices, which are used for storing the values of the BNN parameters, have limited stable states. Consequently, we propose post-training and pre-mapping quantization to allow each of N the samples to be directly implemented on spintronics devices. For this, sampled parameters $\boldsymbol{\theta}$ are quantized to Q-bit values as:

$$\hat{\boldsymbol{\theta}} = \text{round}\left(\text{clip}(\boldsymbol{\theta}, -1, 1) \times \left(2^{Q-1} - 1\right)\right), \tag{3}$$

where $\hat{\boldsymbol{\theta}}$ refers to the Q-bit quantized parameter values, $\text{clip}(\mathbf{x}, \text{min}, \text{max})$ clamps all the elements of input $\mathbf{x}$ within the range $[\text{min}, \text{max}]$, and $\text{round}(\mathbf{x})$ rounds the elements of $\mathbf{x}$ the closest integer. Alternatively, also other quantization functions, such as DoReFa [29], can be used. However, during training, a small prior for the $\boldsymbol{\sigma}_{\omega}^2$ must be chosen as we use $\text{clip}(\mathbf{x}, \text{min}, \text{max})$ post-training.

The specific number of bits the parameters are quantized depends on the representation capabilities of spintronics bit-cell and acceptable performance level. There is a trade-off between the quantization level leading to bit-cell design and performance. We perform design space exploration to determine the optimal quantization level.

Usually, normalization layers standardize neuron distribution during training. Subsequently, quantization can significantly alter neuron output distributions relative to trained distributions. The mismatch between the trained distributions and post-quantized distribution can lead to drastic accuracy degradation. To compensate for the drastic changes in the neuron distributions and bring them close to the training distributions, we propose a *re-calibration*

method. Specifically, the variables in the normalization layers (e.g., mean and variance of BatchNorm) are adjusted. This can be done by estimating them based on several mini-batches under the quantized parameters $\hat{\boldsymbol{\theta}}$ and desired n-way inference. The resulting distribution $\tilde{q}(\boldsymbol{\theta})$ is regarded as an approximation of variational distributions $q(\boldsymbol{\theta})$. Also, since the overall approximation aims to implement them to in-memory architectures, we refer to them as *Bayesian in-memory approximation*.

3.3 Efficient Sampling from CiM Architectures

Although, our proposed in-memory Bayesian approximation allows mapping the posterior distribution to CiM hardware, sampling from them is challenging as well. To this goal, an arbiter per layer is proposed that will select one of the N crossbars in each forward pass. The resulting distribution for the parameters is then $\tilde{q}(\boldsymbol{\theta}) := \text{Choose}(\mathcal{S})$, where the set $\mathcal{S}$ denotes the set of all parameter combinations that can be drawn this way, and Choose$(\cdot)$ denotes a uniform selection from $\mathcal{S}$. $\mathcal{S} = \{\boldsymbol{\theta}^{(s)} \sim q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \mid s = 1, \ldots, S\}$. The overall network topology for the proposed **Spin**tronics based **Bayes**ian (SpinBayes) NNs is depicted in the Fig. 2.

Hence, the inference for $\mathbf{y}^*$ given $\mathbf{x}^*$ is expressed as

$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) \approx \frac{1}{T} \sum_t p\left(\mathbf{y}^* \mid \mathbf{x}^*, \boldsymbol{\theta}^{(t)}\right) \quad \text{with} \quad \boldsymbol{\theta}^{(t)} \sim \tilde{q}(\boldsymbol{\theta}) = \text{Choose}(\mathcal{S}). \tag{4}$$

Note that (1) we may also select $n < N$ crossbars at once for each layer and average their results (n-way inference). (2) the size of $\mathcal{S}$ is different depending on the number of crossbars chosen in the forward pass, in other words, the choice of n in n-way. Generally, for an n-way inference and L layers of crossbars, there are $\binom{N}{n}^L$ possible combinations in $\mathcal{S}$.

3.4 Bayesian Inference on CiM Architecture

To implement the proposed Bayesian posterior distribution, a novel spintronic-based CiM architecture is proposed. The architectural design is shown in Fig. 2 and is based on the network topology in Fig. 1. The spintronic architecture is designed around a multi-value cell crossbar array. The quantized weights are stored in the multi-value cell based on four MTJs, see Fig. 5(a). The number of conductance levels for each cell depends on the number of MTJs. To allow for reliable reading and writing of the cell, four MTJs were used. In fact, each multi-level device possesses five levels of conductance (4AP, 3AP-1P,2AP-2P, 1AP-3P, 4P). To achieve more conductance levels, several multi-level cells could be used jointly for the quantized weight (Q-bit) storage [28].

In each forward pass, a stochastic Arbiter chooses randomly n crossbars per layer in an n-way out of the N available crossbars. Each of the n crossbars receives the same input, here $n = 1$. Two decoders per crossbar allow the activation of multiple addresses and are used for selecting the devices: one for the reading operation and the other one for the writing operation. The spintronic arbiter is connected to the enable signal of the reading decoder of the different crossbar arrays.

The Matrix-Vector Multiplication (MVM) operation in each crossbar is performed by activating multiple wordlines in the array. The resulting current in the bit-line of each crossbar is sensed by a flash analog to digital converter. A CMOS-based Adder-Accumulator (AAC) is utilized to sum up and accumulate different crossbar contributions. Finally, at the output of the crossbar, a CMOS circuit computes layer-wise average weighted sums as mentioned in Section 3.2. The activation functions (not shown in Fig. 2) are implemented with comparators to get the overall activation of the

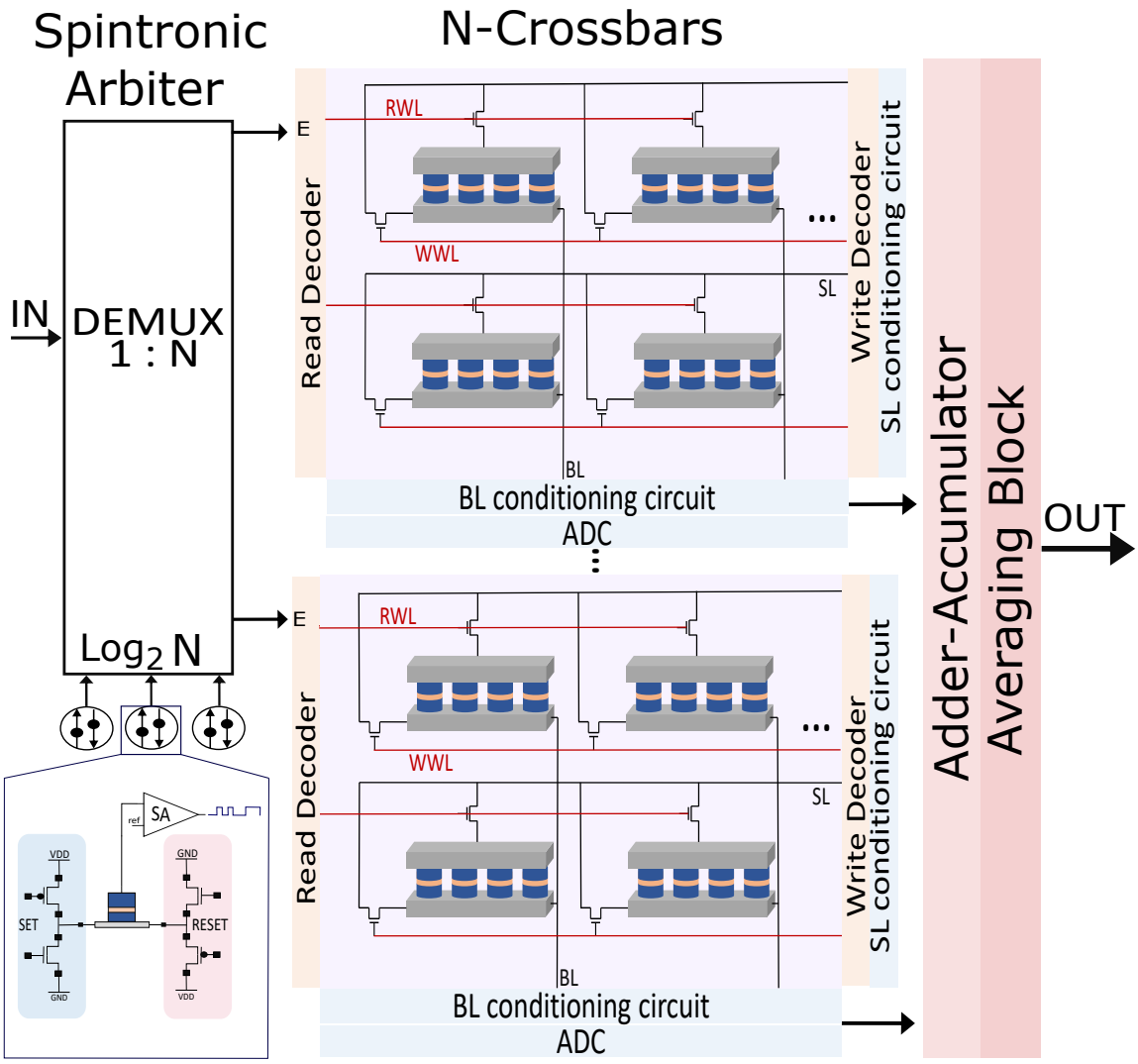


Fig. 2. Proposed layer architecture

layer before feeding it to the subsequent layers. The same averaging circuits are used as the very last layer of the SpinBayes to estimate the prediction of the T stochastic forward passes, as expressed in equation (4).

3.5 Hardware Implementation

3.5.1 Spintronic Arbiter. One spintronic-based arbiter per layer is implemented to allow $n = 1$ in n-way selections of a crossbar in each layer of SpinBayes topology. To this end, the stochastic regime of an MTJ is exploited as a random number generator. The probability density function of switching the SOT-MTJ is expressed as follows [16]:

$$p_{\mathrm{sw}}(I) = 1 - \exp\left[\frac{t}{\tau_0}\exp\left(-\Delta\left(1 - 2\frac{I}{I_{c0}}\left(\frac{\pi}{2} - \frac{I}{I_{c0}}\right)\right)\right)\right] \tag{5}$$

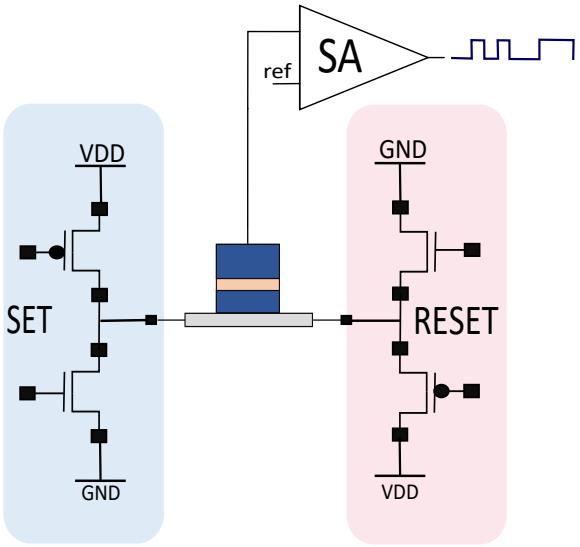


Fig. 3. Spin-Orbit Torque random number generator

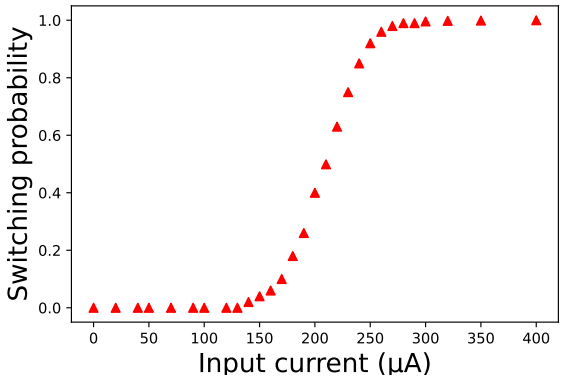
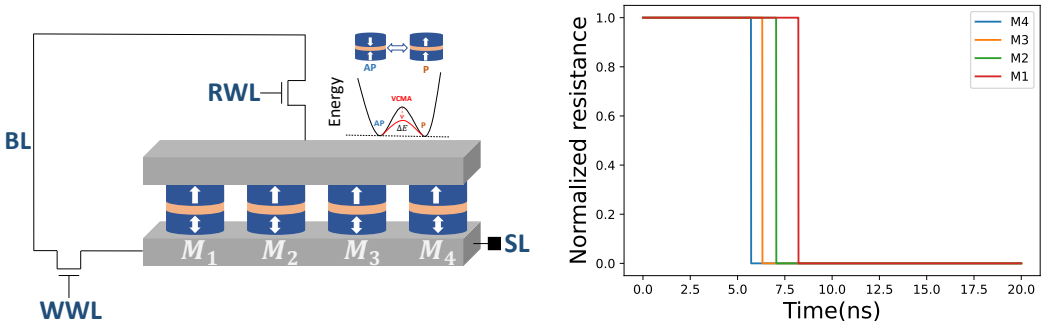


Fig. 4. The probability of switching the stochastic device of the spintronic arbiter

Δ is the thermal stability factor, t is the pulse duration, τ_0 is the attempt time and I_{c0} is the critical current at 0 K. The equation (5) is modeled and evaluated for different switching currents values as depicted in Fig. 4. It is worth noting that in order to have a probability of switching the MTJ of 50%, a current of 230 µA is needed. To generate the required bidirectional current across the SOT-track, four transistors are added, see Fig.3. To ensure a switching probability of 50%, the MTJ is programmed by successive "SET" and "RESET" operations with 230 µA switching current.

To reliably switch the MTJ, the write duration is chosen to be 10 ns for the SET operation and 5 ns for the RESET operation. The state of the MTJ is read using a sense amplifier. The SET and RESET cycles are repeated to generate a stochastic sequence that will be interpreted by a DEMUX (1-to-N) to the enable crossbar's read wordline selection. For the selection of crossbars, $\log_2 N$ stochastic spintronic selection circuits are used to control the DEMUX. The stochastic regime of an MTJ is therefore used here as a random switch selector for the DEMUX, hence for the N Crossbars.



(a) Multi-value cell with 4 MTJs (b) Multi-value cell switching

Fig. 5. Proposed multi-value SOT bit-cell

3.5.2 SOT multi-value cell design for Quantized Weights. The multi-value cell design for the representation of the quantized parameter values is inspired by [8]. Several MTJs were implemented in parallel in order to simulate a multi-level spintronic memristor exploiting the VCMA and SOT effect. The overall cell design is depicted in Fig. 5 (a). The VCMA effect, which depends on the voltage across the MTJ, lowers the switching barrier and eases the writing for a given SOT current. Note that only two access transistors are necessary for read and write operations of the multi-value cell, compared with traditional MTJ implementation in which each MTJ would require two transistors. This results in more efficient use of the hardware resources since the cell is denser. The access transistors are controlled by a Write Word-Line (WWL) and a Read Word-Line (RWL). Choosing a different RWL voltage allows us to program an individual MTJ or multiple MTJs at a time, thus, the versatility of the cell utilization increases. Each MTJ undergoes a different voltage drop in this structure depending on its position on the SOT-track. For instance, M_4 experiences a higher voltage drop than M_1, which results in a reduced switching time for this specific MTJ as shown in Fig. 5 (b). A differential conductance pair scheme increases the number of levels (9 levels) to be considered for the computation, as demonstrated in [8] for ternary implementation. However, a larger quantization range is required to encode the required $Q = 4bits$. Thus, several multi-value cells are used jointly [28]. In crossbar design, the behavior of the same SOT-MTJ is exploited, in a stochastic and deterministic manner. Fig. 4 shows that for a current above 300 µA, the probability of switching the device is close to 1. To use the same device for the arbiter and the weight storage, it is necessary to adapt the writing current of the device. Decreasing the writing current for weight storage might lead to writing failures, which can impact the accuracy of CiM operations.

In the design simulation, a current of 500 µA is used to write the MTJ. The RWL voltage varies from 0 to 1 V with a period of 10 ns. Fig. 5(b) depicts the switching times of the various MTJs (denoted M_x) for an RWL voltage of 1 V. It is worth noting that at this specific voltage, all the MTJs are switched.

For this implementation, the Tunnel Magneto Resistance (TMR) ratio, which represents the Ron/Roff ratio of spintronic devices, is set to 120%. The resistance values are calibrated to have low output current and vary from 2.5 MΩ to 5.5 MΩ. Given the low TMR of spintronic devices, only a limited number of cells per bitline can be reliably sensed in parallel. Thus, only 4 cells can be activated and sensed in parallel, the current sum in the output of the crossbar is then compared to the different references of the flash ADC and sent to the AAC.

4 RESULTS

4.1 Simulation Setup

The predictive performance and uncertainty estimates of the proposed method are evaluated on classification (upto 100 classes) and semantic segmentation tasks (binary as well as multi-class). The semantic segmentation task divides an image into multiple segments and labels each pixel with its corresponding class label. Therefore, it is more difficult compared to classification tasks due to their higher level of granularity.

The LeNet, VGG based on [20], MobileNet, and ResNet topologies (up to 34 layers and 21.79×) are used for the classification tasks that are trained on MNIST, CIFAR-10, and CIFAR-100 as in-distribution datasets ($\mathcal{D}_{in}$). Furthermore, the segmentation tasks were evaluated on two critical tasks: real-world medical image diagnosis with Kvasir-SEG [12] and skin lesion [19], and automotive scene understanding with the CamVid [6] dataset. Kvasir-SEG contains medically obtained gastrointestinal polyps and skin lesion contains microscopic skin lesions. Both medical datasets contain two classes. The CamVid consists of road scene images and has 12 classes. All the segmentation tasks are trained on feature pyramid networks (FPN) [17] with ResNet-18 as the encoder. BNN parameters are quantized 4-bit (Q=4) for all tasks and topology.

Uncertainty evaluation metrics correct-certain ratio (R_{cc}), incorrect-uncertain ratio (R_{iu}), and Uncertainty Accuracy (UA) are based on [20]. The pixel-wise confidence mask for segmentation is calculated by finding the class-wise sum of the normalized variance of T forward passes. The Intersection Over Union (IoU) metric is the ratio between the area where the predicted and ground-truth masks overlap and their union.

For uncertainty estimation, the Out-of-distribution (OoD) detection capability of the models is evaluated on six datasets that came with Pytorch's vision (TorchVision) framework, e.g., (1) Gaussian: random Gaussian noise, (2) Noisy $\mathcal{D}_{in}$: random Gaussian noise added to the test set of in-distribution data, (3) SVHN: Google street view house numbers, (4) FakeData: fake data of randomly generated images, (5) STL10: a dataset containing images from the popular ImageNet dataset, (6) and FashionMNIST: grayscale images of 10 clothing classes. MC-Dropout [9] showed that the prediction probability for OoD data spreads out more compared to $\mathcal{D}_{in}$. Therefore, an OoD is detected if the quantile score of the T forward passes are below 1% and the prediction probability is below 90%. That is to say, an input is rejected if the 99% prediction probabilities for T sub-inferences are below 90%.

4.2 Algorithmic Results

Here, we thoroughly evaluate our proposed approach from both an algorithmic and hardware implementation perspective. However, it is important to note that BNNs are fundamentally different from conventional NNs and that they inherently require more resources to perform the Bayesian inference with the additional benefit of uncertainty information. Therefore, a comparison of the proposed method is only made with the related BNN methods.

4.2.1 Predictive Performance. For both segmentation and classification tasks, our experimental results show that the proposed SpinBayes network performs comparably to State-Of-The-Art (SOTA) methods MC-Dropout [9], MC-Dropconnect [20], and Bayes by Backpropagation (BB-BackProp) [5]. Specifically, relative to the SOTA, performances are generally comparable with up-to 1.14% improvements. For the segmentation tasks, predicted masks are inaccurate around the boundaries of the object classes and obscure classes, e.g., a cyclist in the CamVid dataset, as shown in Fig. 6. For our analysis, we have used $n = 2$ (n-way) inference. However, we have evaluated higher and lower n-way inference. In both cases, the inference accuracies are comparable, e.g., $n = 1$ the pixel-wise accuracy of KvaSir biomedical

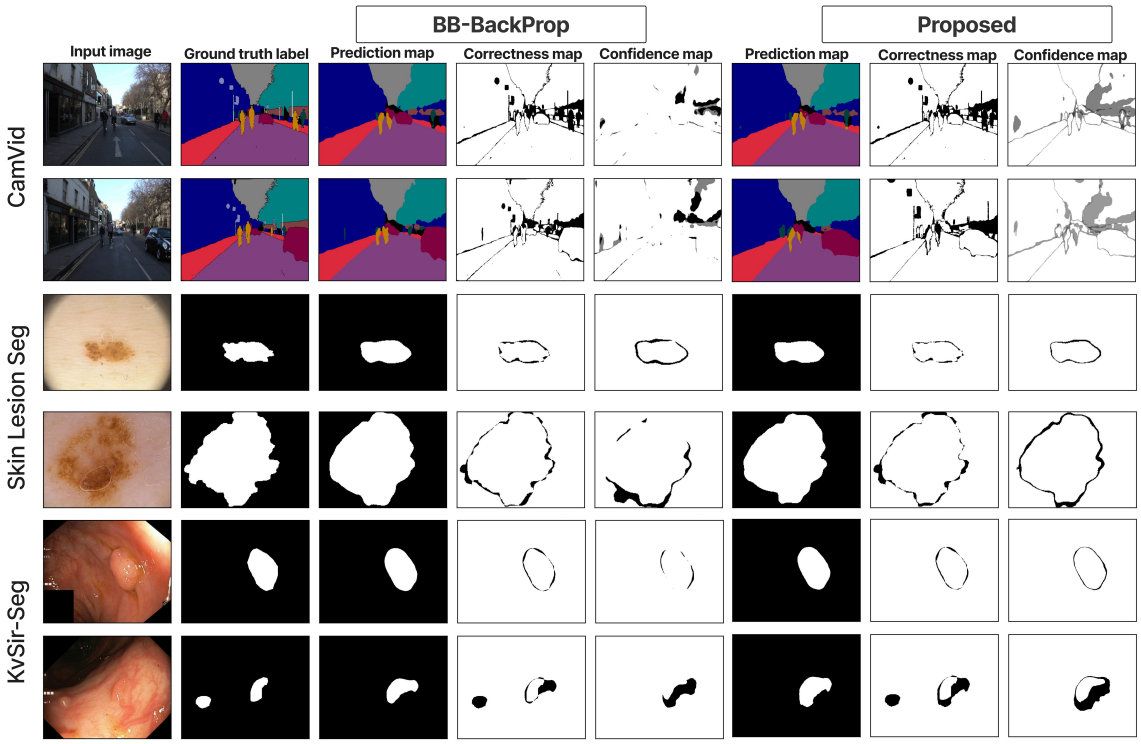


Fig. 6. Qualitative analysis of segmentation tasks. The correctness map shows pixel-by-pixel correctness, with white representing correct predictions and black representing incorrect ones. The confidence map shows pixel-by-pixel uncertainty; white regions are correct and certain.

segmentation tasks goes from 94.93% to 94.56%. Similarly, for $n = 3$, the pixel-wise accuracy does not change. In general, we have found choosing $n > 1$ can improve the accuracy.

4.2.2 Uncertainty Estimation. The uncertainty estimation matrices R_{iu}, R_{cc}, and UA are significantly better compared to MC-Dropout and DropConnect on the same dataset for classification tasks. The proposed method performs comparably in general to BB-BackProp. For example, in the best case, the proposed method improves uncertainty estimation matrices by 20.16%. as demonstrated in Table 1. Furthermore, as shown in Table 3, the proposed method can detect up to 100% of the samples from the out-of-distribution dataset.

However, for uncertainty estimation, 1-way inference ($n = 1$) should be used. Otherwise, more than 40% of the detection capability is reduced. This is because when $n > 1$, the variance for the Bayesian inference is reduced as we do local averaging for each layer, which in turn reduces the uncertainty. Note that, since Fashion-MNIST has a single input channel, it is only evaluated on the MNIST model.

Similarly, for the segmentation tasks, the proposed method performs significantly better compared to MC-Dropout and DropConnect. Also, our result shows similar uncertainty estimation matrices compared to BB-Backprop. Specifically, our method can achieve up to 15.69% better uncertainty estimation matrices, as depicted in Table 2.

Table 1. Analysis of test prediction error and uncertainty estimation performance for MNIST, CIFAR-10, and CIFAR-100 on popular NN topologies.

Dataset	Method	Topology	Prediction error (%)	Uncertainty metrics (%)		
				Riu	Rcc	UA
MNIST	MC-Dropout[9]	LeNet-5	0.77	31.24	98.77	97.48
	MC-DropConnect[20]		0.57	41.67	99.57	98.87
	BB-BackProp[5]		0.75	94.94	99.95	91.99
	Proposed		**0.77**	**96.81**	**99.96**	**85.10**
CIFAR-10	MC-Dropout[9]	VGG Small	10.57	38.24	92.12	82.89
	MC-DropConnect[20]		10.15	40.29	94.31	87.27
	BB-BackProp[5]		10.17	62.43	95.50	88.22
	Proposed		**10.12**	**82.59**	**97.47**	**83.00**
	BB-BackProp[5]	ResNet-34	7.41	78.40	98.03	82.25
	Proposed		**7.43**	**80.82**	**98.17**	**84.00**
	BB-BackProp[5]	MobileNetV2	9.20	83.86	97.88	78.31
	Proposed		**9.26**	**95.44**	**99.22**	**67.79**
CIFAR-100	BB-BackProp[5]	ResNet-18	30.10	53.50	0.98	53.43
	Proposed		**31.06**	**59.54**	**0.91**	**59.33**
	BB-BackProp[5]	ResNet-34	27.48	46.29	0.86	59.48
	Proposed		**27.91**	**54.22**	**1.05**	**54.17**

Table 2. The quantitative prediction and uncertainty estimate performances of the proposed method and state-of-the-art methods.

Data	Method	Prediction Performance (%)			Uncertainty metrics (%)		
		Pixel acc.	Mean acc.	IOU	Riu	Rcc	UA
CamVid	MC-Dropout[9]	80.99	65.46	47.31	17.23	82.48	80.18
	MC-DropConnect[20]	82.92	67.47	49.53	21.63	86.54	82.78
	BB-BackProp[5]	88.07	73.39	49.17	12.36	92.08	82.17
	Proposed	**89.21**	**74.33**	**52.12**	**27.84**	**99.99**	**92.52**
Skin Lesion	BB-BackProp[5]	96.37	96.37	92.05	56.32	83.26	29.24
	Proposed	**96.16**	**96.16**	**91.53**	**50.46**	**98.95**	**32.46**
KvaSir	BB-BackProp[5]	94.64	94.64	80.39	58.54	90.19	20.55
	Proposed	**94.93**	**94.93**	**81.49**	**48.85**	**99.50**	**20.27**

Quantitative observations for the segmentation tasks are depicted in Fig. 6. Ideally, uncertainty around incorrectly predicted pixels should be higher. That is, uncertainty should be higher around black pixels in the correctness mask. Our results show higher uncertainty around incorrect pixels compared to BB-backprop. Therefore, it produces a better uncertainty mask.

4.3 Hardware Results

In this section, we compare the deep ensemble method with state-of-the-art Bayesian architecture proposed in the literature.

4.3.1 Energy Consumption Analysis. The energy consumption of different elements of the proposed architecture is presented in Table 4. The AAC, the comparator, and the averaging circuits were synthesized with Synopsys Design

Table 3. Analysis of the Proposed Method for Detecting Out-of-Distribution Data (OoD).

$\mathcal{D}_{in}$	Topology	OoD Detection (%)					
		Gaussian	Noisy $\mathcal{D}_{in}$	SVHN	FakeData	STL10	FashionMNIST
MNIST	LeNet-5	100.0	99.21	100.0	100.0	100.0	100.0
CIFAR-10	VGG-Small	99.86	94.35	80.33	86.89	89.31	-
	ResNet-34	99.99	98.14	99.99	100.0	99.94	-
	MobileNetV2	98.40	95.31	98.80	97.54	98.06	-
CIFAR-100	ResNet-18	99.99	91.17	98.96	99.45	99.31	-
	ResNet-34	94.45	86.29	68.18	70.01	79.6	-

Table 4. Architecture level estimations

Circuit	Dynamic energy	Circuit	Dynamic energy
Memory (Decoding/Sensing)	4.71 pJ	Adder-Accumulator	0.06 pJ
Spintronic RNG	3.80 pJ	Comparator	0.01 pJ
Averaging block	18.42 pJ		

Compiler, using TSMC 40 nm PDK. The Spintronic arbiter and the memory operations were evaluated at the circuit level. Furthermore, to assess the energy consumption of a BNN given a LeNet-5 topology, We estimated the number of computational operations required for each layer (e.g., convolution, pooling, etc.). Given the number of operations performed in parallel in the crossbar architecture along with the dynamic energy evaluation presented in Table. 4. We utilized a modified NVSim simulator for CiM to scale the architecture and estimate the overall power consumption for one inference.

In Table 5, the energy consumption with 10 forward passes is compared against other FPGA and CiM implementations. For a fair comparison, we used the same metrics and dataset (MNIST) as existing works. We differ only in terms of the topology (LeNet-5 CNN topology). We achieve an energy efficiency of 80× better when compared to FPGA implementation [4], 35× better when compared to RRAM [18], and 3× better when compared to an MTJ-based crossbar [27]. Note that the above-referenced implementations involved a smaller network consisting of two linear layers (200 neurons).

Also, the proposed implementation requires also fewer RNGs, for instance, work in [18] encoded all the inputs of the crossbar with Gaussian RNG based on RRAM variability. Work in [27] used an array of 4 devices interfaced with an accumulator to implement the Gaussian RNG. Work in [1] used also several stochastic STT-MRAM devices per wordline to implement dropout[9].

As a result, SpinBayes reduces the computational overhead associated with generating random numbers by proposing an arbiter with only 3 RNGs per layer.

Table 5. Energy comparison with SOTA implementation

Method	Implementation	Number of RNG	Energy (µJ/Image)
H.Awano et al. [4]	FPGA	-	21.09
A. Malhotra [18]	RRAM	64	9.30
K.Yang et al.[27]	Domain wall-MTJ	Array (2×2)	0.79
Proposed implementation	**SOT-MRAM**	3	**0.26**

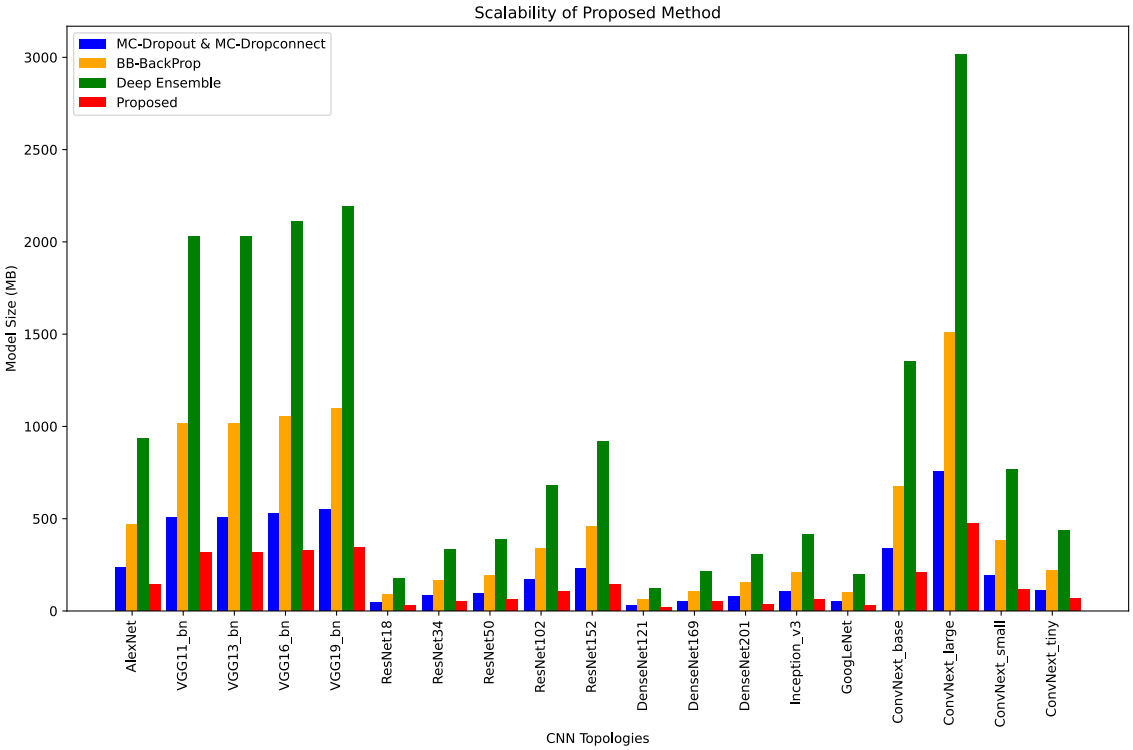


Fig. 7. The scalability of the proposed method is evaluated in terms of model size (in MB) for different large CNN topologies. The model size for each of the topologies is compared to SOTA BNN methods, namely MC-Dropout & MC-Dropconnect, BB-BackProp, and Deep Ensemble. The proposed method achieves a significantly smaller model size compared to the other BNN methods across all CNN topologies.

4.3.2 Memory Overhead Analysis. Resource requirements are an inherent drawback of most BNNs and ensemble methods. Therefore, the scalability of BNNs to larger topologies is challenging. Since we have used low-bit precision quantization (4-bit), a common model compression technique, our overall memory consumption is significantly lower.

Specifically, Our proposed architecture generates significantly lower memory overhead compared to existing BNN implementations. Specifically, it requires 8×, 1.6×, and 6.39× less memory compared to the ensemble method [15] with 5 ensembles, MC-methods [9, 20], and BB-BackProp [5], respectively, on the topologies we have evaluated (see Table 1 and 2). We assumed SOTA methods are full-precision 32-bit and require a single cell to store each bit.

Similarly, our proposed approach is scalable for even larger topologies. As can be seen in Fig. 7, the proposed method achieves the smallest model size for all the CNN topologies. The topologies evaluated here can be found in PyTorch's vision library.

4.4 Discussion

n-ways Inference. In our proposed design, only one crossbar ($n = 1$) is randomly selected for an n-way inference. First, selecting one crossbar reduces runtime power consumption since one memory unit is used at a time. Furthermore, activating and accessing multiple crossbars simultaneously increases the periphery overhead and complexity. Moreover,

this choice is also motivated algorithmically. Although, ($n > 1$) increases the accuracy only slightly, the reduction in OoD detection capabilities is significant. Therefore, $n = 1$ is recommended as the main goal of our approach is low uncertainty estimates without sacrificing performance.

Model Compression. When comes to choosing bit-precision for the parameters, there is a trade-off between accuracy and hardware overhead. A higher bit-precision parameter would improve the accuracy but implementing a high bit-precision requires challenging sensing schemes and hardware overhead due to complex ADCs. Furthermore, most NVM devices achieve only a few conductance levels [26]. Thus, for an efficient CiM implementation, sufficiently lower bit quantization is required. In our analysis and circuit design, we have chosen 4-bit precision to represent parameters, as it offers the best trade-off between accuracy and hardware overhead.

Although, we have shown that our method is significantly smaller in terms of memory requirements, if even further compression is required, unimportant neurons can be removed (pruning) [11]. However, such a method usually requires sensitivity analysis. Also, since the size of the memory array is fixed, pruning may not beneficial as it may lead to underutilized memory cells. CiM architecture-aware pruning can be performed [13], but such kind of evaluation is beyond the scope of the paper.

Number of Monte-Carlo Runs for the Bayesian Inference. The number of Monte Carlo runs required for a BNN can significantly increase the runtime power consumption and latency per inference results. We have found that increasing the Monte Carlo runs beyond some point does not increase the accuracy, but only increases the runtime power and latency.

5 CONCLUSION

We present SpinBayes, a Bayesian inference approach suitable for SOT-based CiM hardware accelerators with a holistic perspective. We propose a multilevel SOT-based bitcell to map quantized parameters for Bayesian inference. In addition, we develop a spintronics-based arbiter that utilizes the inherent stochasticity of spin-orbit torque devices to select crossbar arrays randomly for each forward pass. On various classification and segmentation tasks, our proposed architecture was rigorously examined for its prediction performance and uncertainty quantification. It enhances prediction performance by up to $\sim$ 1%, can detect up to 100% of various out-of-distribution data, and generates superior uncertainty masks for segmentation tasks. In addition, the memory overhead is smaller by 8× and the energy consumption by 80× compared to state-of-the-art CMOS implementations.

REFERENCES

[1] Soyed Tuhin Ahmed, Kamal Danouchi, Christopher Münch, Guillaume Prenat, Lorena Anghel, and Mehdi B. Tahoori. 2023. SpinDrop: Dropout-Based Bayesian Binary Neural Networks with Spintronic Implementation. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2023), 1–1. https://doi.org/10.1109/JETCAS.2023.3242146

[2] Mustafa Ali, Sourjya Roy, Utkarsh Saxena, Tanvi Sharma, Anand Raghunathan, and Kaushik Roy. 2022. Compute-in-Memory Technologies and Architectures for Deep Learning Workloads. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30, 11 (2022), 1615–1630. https://doi.org/10.1109/TVLSI.2022.3203583

[3] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565* (2016).

[4] Hiromitsu Awano et al. 2020. BYNQNet: Bayesian Neural Network with Quadratic Activations for Sampling-Free Uncertainty Estimation on FPGA. In *DATE*.

[5] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. 2015. Weight Uncertainty in Neural Network. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 37)*, Francis Bach and David Blei (Eds.). PMLR, Lille, France, 1613–1622. https://proceedings.mlr.press/v37/blundell15.html

[6] Gabriel J Brostow, Julien Fauqueur, and Roberto Cipolla. 2009. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters* (2009).

[7] B. Dieny et al. 2020. Opportunities and challenges for spintronics in the microelectronics industry. *Nature Electronics* 3, 8 (Aug. 2020). https://doi.org/10.1038/s41928-020-0461-5

[8] J. Doevenspeck, K. Garello, S. Rao, F. Yasin, S. Couet, G. Jayakumar, A. Mallik, S. Cosemans, P. Debacker, D. Verkest, R. Lauwereins, W. Dehaene, and G.S. Kar. 2021. Multi-pillar SOT-MRAM for Accurate Analog in-Memory DNN Inference. In *2021 Symposium on VLSI Technology*. 1–2.

[9] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*. PMLR, n.d., 1050–1059.

[10] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, et al. 2021. A survey of uncertainty in deep neural networks. *arXiv preprint arXiv:2107.03342* (2021).

[11] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[12] Debesh Jha, Pia H Smedsrud, Michael A Riegler, Pål Halvorsen, Thomas de Lange, Dag Johansen, and Håvard D Johansen. 2020. Kvasir-seg: A segmented polyp dataset. In *MultiMedia Modeling: 26th International Conference, MMM 2020, Daejeon, South Korea, January 5–8, 2020, Proceedings, Part II 26*. Springer, n.d., 451–462.

[13] Biresh Kumar Joardar, Janardhan Rao Doppa, Hai Li, Krishnendu Chakrabarty, and Partha Pratim Pande. 2022. ReaLPrune: ReRAM Crossbar-Aware Lottery Ticket Pruning for CNNs. *IEEE Transactions on Emerging Topics in Computing* (2022).

[14] Melody Y. Kiang. 2003. Neural Networks. In *Encyclopedia of Information Systems*, Hossein Bidgoli (Ed.). Elsevier, New York, 303–315. https://doi.org/10.1016/B0-12-227240-4/00121-0

[15] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. *NeurIPS* (2017).

[16] Seo-Won Lee and Kyung-Jin Lee. 2016. Emerging Three-Terminal Magnetic Memory Devices. *Proc. IEEE* 104, 10 (Oct. 2016), 1831–1843. https://doi.org/10.1109/JPROC.2016.2543782 Conference Name: Proceedings of the IEEE.

[17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2017. Feature pyramid networks for object detection. In *CVPR*.

[18] Akul Malhotra et al. 2020. Exploiting Oxide Based Resistive RAM Variability for Bayesian Neural Network Hardware Design. *IEEE TNANO* (2020).

[19] Teresa Mendonça, Pedro M Ferreira, Jorge S Marques, André RS Marcal, and Jorge Rozeira. 2013. PH 2-A dermoscopic image database for research and benchmarking. IEEE, 5437–5440.

[20] Aryan Mobiny, Pengyu Yuan, Supratik K Moulik, Naveen Garg, Carol C Wu, and Hien Van Nguyen. 2021. Dropconnect is effective in modeling uncertainty of bayesian deep networks. *Scientific reports* (2021).

[21] Radford M Neal. 2012. *Bayesian learning for neural networks*. Springer Science & Business Media.

[22] Dhinakaran Pandiyan and Carole-Jean Wu. 2014. Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 171–180.

[23] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. 2021. AI accelerator survey and trends. In *IEEE HPEC*.

[24] Qiming Shao et al. 2021. Roadmap of spin-orbit torques. *IEEE TransMag* (2021).

[25] Mattias Teye, Hossein Azizpour, and Kevin Smith. 2018. Bayesian uncertainty estimation for batch normalized deep networks. In *ICML*. PMLR.

[26] Zhongrui Wang, Huaqiang Wu, Geoffrey W Burr, Cheol Seong Hwang, Kang L Wang, Qiangfei Xia, and J Joshua Yang. 2020. Resistive switching materials for information processing. *Nature Reviews Materials* 5, 3 (2020), 173–195.

[27] Kezhou Yang et al. 2020. All-Spin Bayesian Neural Networks. *IEEE T-ED* (2020).

[28] Shimeng Yu. 2018. Neuro-inspired computing with emerging nonvolatile memorys. *Proc. IEEE* 106, 2 (2018).

[29] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. (2016).

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009