Accelerating Bayesian Neural Networks on Low-Power Edge RISC-V Processors

Samuel Pérez[1], Javier Resano[1] and Darío Suárez Gracia[1]
{779333, jresano, dario}@unizar.es

***Abstract*— Neural Networks (NNs) are a very popular solution for classification tasks. As the combination of Internet of Things (IoT) with Machine Learning (ML), also known as TinyML, grows in popularity, more NN are being executed on low-end edge systems. The reliability of the predictions is crucial for safety-critical applications. Bayesian Neural Networks (BNNs) address this issue by calculating uncertainty metrics with their predictions at the cost of increasing computing requirements. This work addresses the challenges of executing BNNs inference on low-end systems. BNNs require multiple forward passes in which the weights are sampled from distributions. This sampling process can take up to 85,13% of execution time. This work optimizes the weight sampling and integrates it within a low cost custom extension for a RISC-V CPU, improving speedup up to ×8,10 and similar energy savings.**

I. Introduction

As the Internet of Things (IoT) ecosystem expands, the need for intelligent systems capable of making informed decisions in the presence of low quality or noisy data is crucial. Understanding and quantifying uncertainties are paramount in the context of IoT systems, where data may be incomplete, noisy, or subject to rapid fluctuations.

Bayesian Neural Networks (BNNs) are a type of Neural Networks that integrate probabilistic modeling, enabling them to quantify uncertainty in machine learning tasks. They can provide uncertainty metrics alongside their predictions improving its confidence and reliability [1]. However, they need more parameters to define the weights (typically twice as many as a conventional NN does), and the inference is more complex. The inference algorithm for BNN requires multiple forward passes of the network, drastically increasing the inference cost.

BNNs sample probability distributions, normally Gaussian distributions, to generate the weights for the forward passes. In a low-power processor, this step has the highest computational cost of the whole inference process. Fig. 1 shows that, on average, more than 86% of the total execution cycles are dedicated to weight distribution sampling in different models. Optimizing this step is a key aspect to enable the BNN inference execution in IoT systems.

This work analyzes the execution of BNNs in resource-constrained IoT environments. Our main objective is obtaining uncertainty metrics at real-time because they are crucial to improve the reliability of the predictions, and help to the decision-making process. We used an in-house integer-only single core RISC-V CPU as our test-bench, because it is a good example of a low-cost IoT CPU. The experiments were run using models trained with TensorFlow Probability (TFP) BNN libraries, and a custom-made C library for integer-only BNN inference for low-power devices.

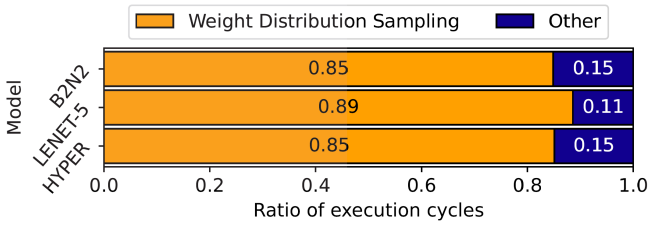


Fig. 1: Cycles breakdown of integer-only inference for multiple Bayesian Neural Network models.

We propose a software weight sampling optimization method for BNN inference along with a study of its impact on the quality of the model outputs. The method takes into account both the accuracy changes and the possible degradation in the uncertainty metrics compared to the baseline models. Using this optimization we obtained up to ×5.88 inference speedup. However, this solution does not scale well, and only achieves accurate results in small models due to error accumulation in larger ones. For this reason, we also propose adding a hardware Gaussian random number generator (GRNG) to a RISC-V CPU, as a new functional unit, and using the custom RISC-V extension framework to add two new instructions that use the GRNG.

These new instructions greatly improve the efficiency of the BNNs inference process as they take care of the distribution sampling step without degrading the model outputs accuracy and uncertainty metrics. Using this extension we obtained up to ×8.10 inference speedup over the baseline models, and almost the similar improvements in energy consumption, since the power-consumption overhead is only 0.65%.

Section II presents the related works on BNN acceleration. Section III provides more details on BNNs and uncertainty metrics. Section IV presents the experimental design, the BNN models and the technologies used. Sections V and VI introduce our software weight sampling optimization and our hardware functional unit, and analyse their results. Finally, Section VII summarizes the key findings and future research directions.

This work was supported by MCIN/AEI/10.13039/501100011033 (grant PID2022-136454NB-C22 and PDC2023-145851-I00), and by Government of Aragon (T58 23R research group).

[1]Department of Computer Sciences and Systems Engineering (DIIS), Aragon Institute of Engineering Research (I3A), University of Zaragoza, HiPEAC European NoE.

II. Related Work

ML acceleration is a very active research field with many recent proposals [2]. BNNs are no exception, given their high inference cost. Previous research has focused on developing high-performance accelerators for the complete inference process [3]–[5]. This work uses a different approach, our goal is to find a way to work with these complex models efficiently on systems with very few computational resources and low power constraints.

Hiromitsu *et al.* proposed a sampling-free accelerator for the BNN inference algorithm replacing the ReLU activation function by a quadratic non-linearity function [4]. They reported good results for the MNIST dataset, but other works have shown that this approach is not generalisable, and does not perform well with other models [5]. For this reason, we will focus on the multiple forward passes approach, as it is the most standard method, and its performance has been extensively demonstrated [1], [3], [5].

Weight sampling is also a critical part of the previous works that follow the multiple forward passes approach. Awano *et al.* approximates the final model output using a hardware Bernoulli sampler [5], implicity assuming that the Central Limit Theorem (CLT) could be applied to the model as a whole. A key advantage of Bernoulli samplers lies in their hardware implementation. Unlike software methods that rely on branching operations, which can degrade the performance of modern CPUs, Bernoulli samplers can be built using simple hardware components like comparators and multiplexers which makes them low cost and efficient. However, in our experiments, the Bernoulli distribution did not achieve satisfactory results in preserving uncertainty metrics and accuracy. Therefore, we have explored an alternative approach using a software uniform sampling algorithm.

Cai *et al.* used two different GRNGs for their inference accelerator [3]. One of these GRNGs is based on the CLT using the Binomial distribution. In this work we will use a smaller CLT-based GRNG based on the sum of Uniform distributions because it reduces computations needed. GNRGs have been explored in depth [6] and previous work has shown that CLT-based GRNGs do not show the highest tail accuracy [7]. However, we have evaluated it for several BNNs, and its accuracy is sufficient to obtain similar results to the original models.

III. Bayesian Neural Networks

Bayesian approaches use the Bayes Theorem, shown in Equation 1, to model the probability of a set of weights w given a training dataset $D = \{x, y\}$. This probability distribution is called the posterior.

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)} \tag{1}$$

The training of a BNN consist of calculating the posterior distribution [8]. Calculating this distribution for a big and complicated model like a NN is intractable. In order to solve this issue, variational Bayesian methods are used to approximate this distribution. The most common method is variational inference, which approximates the posterior distribution using a variational distribution $q_\phi(w)$. This approximation is done by minimizing the Kullback-Leibler of $q_\phi(w)$ and $p(w|D)$ updating a set of parameters ϕ. Theoretically, the distributions could have any shape, but to reduce the search space only symmetric and tractable distributions are used. Gaussian distributions are a good option because they are only defined with two parameters which helps in reducing the size of the model.

A trained BNN model consists of a set of means and standard deviations of different Gaussian distributions $\phi = \{\mu, \sigma\}$, meaning that weights become probability distributions instead of fixed values, and the output also becomes a distribution instead of a single value. This enables to measure the uncertainty of the prediction. Equation 2 shows the posterior prediction distribution of a new observation x^*.

$$p(y^*|x^*, D) = \int p(y|x^*, w)p(w|D)dw \tag{2}$$

Monte Carlo (MC) sampling can approximate the integral of Equation 2. The samples in this case are different stochastic forward passes. In each of these passes, the weights are sampled from the Gaussian distributions that form the approximated posterior.

A. Measuring Uncertainty

In a classification setting, we can measure the uncertainty of a prediction using the MC samples [9]. Let T be the number of MC samples, K the number of classes of the dataset and a sample $a_t = p(y = c_k|x, w_t)$. The predictive uncertainty ($\mathbb{H}$) represents the uncertainty of a prediction in the range $[0, \log(K)]$ and can be calculated using Equation 3.

$$\mathbb{H}(y|x, D) = -\sum_{k=1}^{K}\left[\left(\frac{1}{T}\sum_{t=1}^{T} a_t\right)\log\left(\frac{1}{T}\sum_{t=1}^{T} a_t\right)\right] \tag{3}$$

The uncertainty metrics enable to identifying predictions that do not reach the required level of accuracy or detecting labeling conflicts in the dataset or input-data corruption. Uncertainty can be divided in two types, aleatoric and epistemic, $\mathbb{H}$ captures both. Expected entropy ($\mathbb{E}_p$) captures the aleatoric uncertainty, which is caused by ambiguities in the dataset such as noisy measurements, overlapping classes, or mislabeled samples. $\mathbb{E}_p$ can be calculated using Equation 4.

$$\mathbb{E}_{p(w|D)}[\mathbb{H}(y|x, D)] = \frac{1}{T}\sum_{t=1}^{T}\left(-\sum_{k=1}^{K} a_t \log(a_t)\right) \tag{4}$$

Epistemic uncertainty represents what the model does not know due to the lack of information during the training process, and it can be calculated using the mutual information (MI) shown in Equation 5.

$$MI(y, w|x, D) = \mathbb{H}(y|x, D) - \mathbb{E}_{p(w|D)}[\mathbb{H}(y|x, D)] \tag{5}$$

IV. Methodology

The proposal validation includes three different BNN architectures to cover a broad range of model sizes for IoT applications. First, we have used a set of small fully-connected BNN models, from [1], designed for pixel classification of five different hyperspectral images (BO, IP, KSC, PU, SV). Table I presents their general architecture. Second, we have used a Bayesian Convolutional Neural Network (CNN), based on the classical Lenet-5 model, for the classification of the MNIST and the CIFAR10 datasets. Its details are described on Table II. Finally, we have used a larger Bayesian CNN model, B2N2, from [5], for the same datasets. Table III shows its architecture details.

TABLE I: Model architecture for hyperspectral pixel classification

Layer Type	Input	Output
Dense	Number of spectral bands	32
Dense	32	16
Dense	16	Number of pixel classes

TABLE II: Bayesian Lenet-5 architecture

Layer Type	Input	Output	Kernel Size
Conv2D Valid	28×28	24×24×6	5×5
MaxPool2D	24×24×6	12×12×6	2×2
Conv2D Valid	12×12×6	8×8×16	5×5
MaxPool2D	8×8×16	4×4×16	2×2
Conv2D Valid	4×4×16	1×1×120	4×4
Dense	120	84	
Dense	84	10	

TABLE III: Bayesian B2N2 architecture

Layer Type	Input	Output	Kernel Size
Conv2D Same	32×32	32×32×32	3×3
Conv2D Same	32×32×32	32×32×32	3×3
MaxPool2D	32×32×32	16×16×32	2×2
Conv2D Same	16×16×32	16×16×64	3×3
Conv2D Same	16×16×64	16×16×64	3×3
MaxPool2D	16×16×64	8×8×64	2×2
Conv2D Same	8×8×64	8×8×128	3×3
Conv2D Same	8×8×128	8×8×128	3×3
MaxPool2D	8×8×128	4×4×128	2×2
Dense	2028	10	

We trained the models in 32-bit floating-point (FP) precision using TesorFlow Probability libraries. But there are no tools to port these models to low-power embedded processors without floating-point support. Hence, we developed tools to do it automatically. First, the models and its weights were transformed to fixed point values and C functions using a custom built library. Then, the inference was performed using our own C code library for integer-only BNN inference.

We used an in-house 32 bit RISC-V CPU with RV32I and multiplication support written in VHDL as our test-bench. The RTL code can be simulated to obtain accurate cycle count metrics using the standard RISC-V hardware counters. This code is fully synthesisable and has been implemented and tested using Xilinx ZCU104 FPGA board and Xilinx Vivado 2023.2. The aforementioned code is publicly available in a GitHub repository [10].

V. Weight Sampling Optimization

As Fig. 1 shows, the weight distribution sampling process takes the most part of the execution cycles. Reusing the same samples for batched inputs could reduce the sampling delay; however, batching significantly increases memory requirements, which invalidates this option for devices with limited memory such as IoT ones. Therefore this work will focus on optimizing the sampling process for online inference.

BNN inference performs multiply accumulation operations (MAC) where the neurons inputs are multiplied by the model weights and then accumulated. This accumulated value follows Gaussian distribution due to the CLT [5]. Eq. 6 and 7 show the expected value and variances of the result of the MAC operation, O, where N, x_i, w_i, and b represent the number of neuron inputs, the input values, the weight distributions and a constant bias, respectively.

$$E[O] = b + \sum_{i=0}^{N} (E[w_i]E[x_i]) \tag{6}$$

$$V[O] = \sum_{i=0}^{N} (V[w_i]V[x_i] + V[w_i]E[x_i]^2 + V[x_i]E[w_i]^2) \tag{7}$$

As long as $E[O]$ and $V[O]$ are preserved along the network, the distribution of O will always be Gaussian. To reduce the delay of the sampling process, we propose sampling, scaling, and displacing a uniform distribution to match the variance and expected value of a Gaussian distribution. While sampling a Gaussian distribution using a simple GRNG CLT based algorithm requires 95 cycles in our CPU, the sampling implementation of a uniform distribution only requires 6 cycles using a 32 bit Xorshift algorithm [11].

Digging into the details, considering an original weight $w \sim \mathcal{N}(\mu, \sigma)$, we can define a new weight $w' = bu + a$, u being a sample from a uniform distribution. a and b are calculated solving the system of equations formed by $\mu = E[w']$ and $\sigma^2 = V[w']$. Equations 8 and 9 show how to obtain a and b from μ and σ.

$$a = \mu - \frac{b}{2} \tag{8}$$

$$b = \sigma\sqrt{12} \tag{9}$$

After training the BNN, all the expected values and variances of the normal distributions are transformed into the new coefficients. During inference, a sample from a uniform distribution is transformed using these coefficients.

Table V shows the inference speedup in the different models. Table IV shows the accuracy comparison between the different generation methods: the baselines are the original TFP Python models. CLT are the results obtained using our BNN integer-only inference using a software CLT GRNG, whereas, Uniform are the results obtained when sampling a uniform distribution to speedup the computations.

TABLE IV: Model accuracy comparison

	Generation Method		
Model	*Baseline*	*CLT*	*Uniform*
Hyperspectral BO	0.9039	0.9101	0.9083
Hyperspectral IP	0.8139	0.8148	0.8131
Hyperspectral KSC	0.9256	0.9217	0.9217
Hyperspectral PU	0.9017	0.9021	0.9017
Hyperspectral SV	0.9257	0.9275	0.9280
Lenet-5 MNIST	0.9836	0.9830	0.8906
Lenet-5 CIFAR10	0.6351	0.6229	0.4436
B2N2 MNIST	0.9872	0.9854	0.0964
B2N2 CIFAR10	0.7295	0.7134	0.2873

We have used 100 MC samples. Due to the quantization and the probabilistic nature of the models, an exact accuracy match is not possible. However, the models using the CLT-based approach provide similar results to the baseline models. Regarding the optimization method based on the Uniform distribution, it reduces execution time 5× while preserving both the accuracy and the uncertainty for the small hyperspectral models. However, accuracy decreases for the larger models and, in the worst cases, it becomes unacceptably low. Probably, the reason is that the accumulation of small distribution errors becomes too large as the number of layers increases.

In BNN models, besides accuracy, uncertainty metrics must be preserved as well. Fig. 2 shows uncertainty metrics comparison diagrams of the KSC model predictions. Other images provided similar results, and are omitted to save space. Subfig. 2a shows the average $\mathbb{H}$ and $\mathbb{E}_p$ of the predictions grouped by class. Subfig. 2b shows the $\mathbb{H}$ histogram of the correct predictions in green and the incorrect predictions in red. The diagrams show that the uncertainty metrics are very similar in both cases. Hence, this optimization maintains the quality of the uncertainty metrics for the small hyperspectral pixel classification models.

LENET-5 MNIST model has a considerable accuracy drop but still keeps a 89.06% accuracy which can be acceptable in some scenarios. However, the uncertainty diagrams shown in Fig. 2 reveal that the uncertainty metrics are completely different from those of the baseline. Therefore, in this case, the uncertainty metrics obtained using uniform distributions cannot be trusted and one of the key advantages of using a BNN is lost.

VI. Custom RISC-V Extension

To reduce the execution time even further, and design a solution that can be used for larger models, we propose adding a small and low cost GRNG to a RISC-V RV32I CPU as a new functional unit. The GRNG will be based on the CLT using a sum of uniform distributions. The relation between a sum of uniforms and a Gaussian distribution in shown in Eq. 10.

$$\sum_{n=1}^{N} \mathcal{U}_n(0,1) \sim \mathcal{N}\left(\frac{n}{2}, \sqrt{\frac{n}{12}}\right) \tag{10}$$

Using 12 uniform distributions we obtain an approximation of $\mathcal{N}(6,1)$, which can be centered with an addition and then scaled and displaced to approximate any Gaussian distribution. We designed a GRNG generator that follow this approach.

To generate the uniform samples we used a Linear Feedback Shift Register (LFSR) [12]. These generators are implemented using a shift register and a feedback network. This network implements a set of XOR operations known as the generating polynomial, the bits used in this operations are known as taps. Given a shift register of n bits, if the generating polynomial provides $2^n - 1$ values before repeating the sequence then the sequence is known as a maximal length sequence. A list of generating polynomials for maximal length sequences can be found at [13].

We used 12 samples of 12 bits, which requires generating 144 bits. To reduce the correlation between the bits we used the skip-ahead technique from [12], which consists in using a more complex feedback network that performs more than one step of the generating polynomial operation each clock cycle. For those reasons we decided to use a 151 bit LFSR with a 144 skip-ahead feedback network.

The uniform samples are then added using a tree of adders with 4 levels and then centered yielding a final 16 bit approximated Gaussian sample. Figure 4 shows a diagram of the proposed GRNG. Since the design is fully-pipelined, it can provide one Gaussian sample per cycle.

We added this GNRG into our RISC-V CPU alongside two new instructions to its instruction set.

1) `genum rd`. This instruction uses the GRNG and stores a gaussian sample in the upper 16 bits of the register `rd`.
2) `setseed ra, rb`. This instruction allows the initialization of the internal registers of the GRNG.

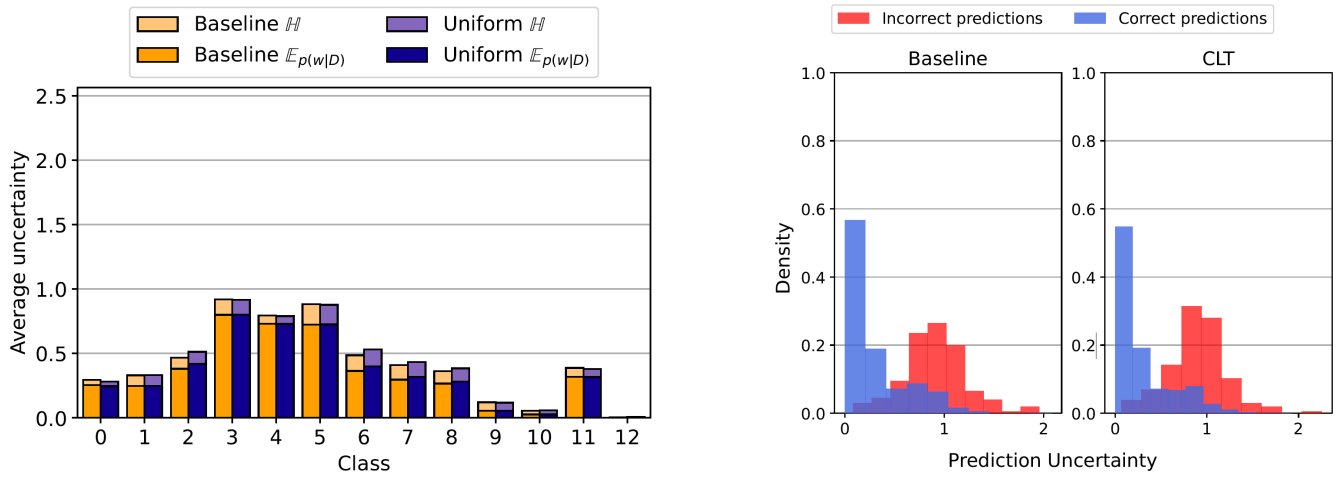
We modified the GNU GCC RISC-V Toolchain to use the new instructions in C code through assembly inlines. With these instructions we obtain similar results to those presented on Table IV with the CLT method, and also similar uncertainty distributions to the baseline models as shown in Fig. 2. In addition, we obtain better speedups than with the optimization based on the uniform distribution, presented on Table V.

TABLE V: Inference cycle cost for the test models on the RISC-V CPU with different optimizations.

	1 forward pass kilocycles			Speedup over CLT	
Model	*CLT*	*Unif.*	*Custom Ins.*	*Unif.*	*Custom Ins.*
BO	600	121	95	4.95	6.35
IP	800	161	125	4.96	6.38
KSC	708	142	110	4.98	6.42
PU	442	90	70	4.92	6.31
SV	814	164	127	4.96	6.38
Lenet-5	30202	5136	3728	5.88	8.10
B2N2	3833300	785336	614102	4.88	6.24

Fig. 5 displays how the proposed optimizations reduce the percentage of inference cycles dedicated to weight sampling. The uniform sampling optimization reduces this percentage from an average 86.20% to 28.57%, and the custom instructions further reduces it to 6.28%.

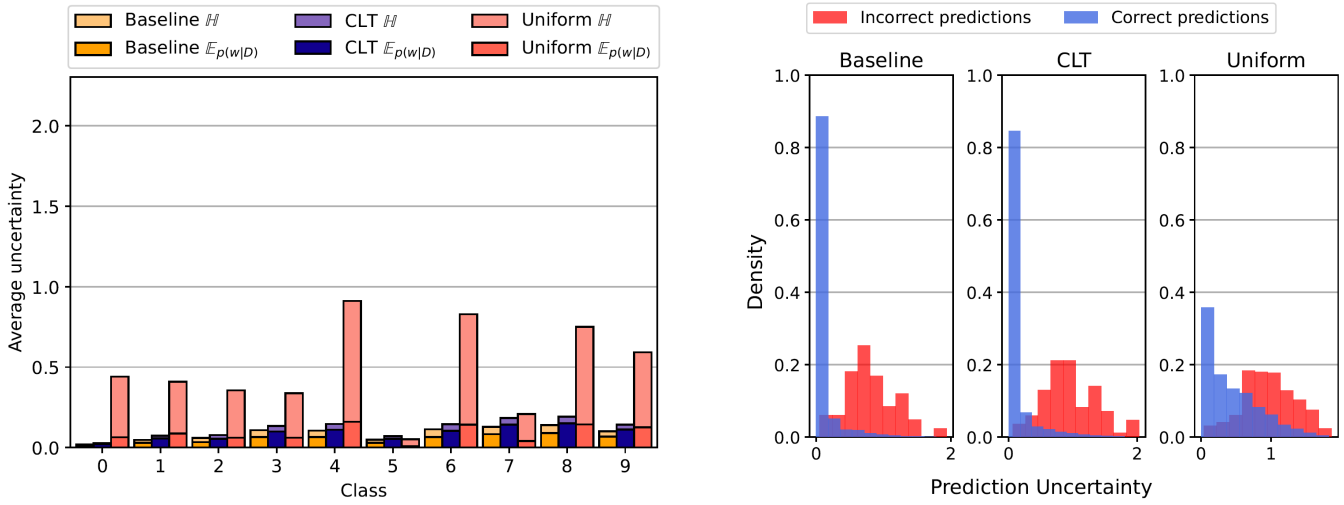
Table VI shows the FPGA resource utilization of the baseline CPU and the extra cost related to the GRNG



(a) Average uncertainty grouped by class

(b) $\mathbb{H}$ histogram of correct and incorrect predictions

Fig. 2: Uncertainty metrics comparison diagrams of the KSC model using different weight sampling algorithms.



(a) Average uncertainty grouped by class

(b) $\mathbb{H}$ histogram divided of correct and incorrect predictions

Fig. 3: Uncertainty metrics comparison diagrams of the LENET-5 model on the MNIST dataset using uniform weight sampling algorithm.

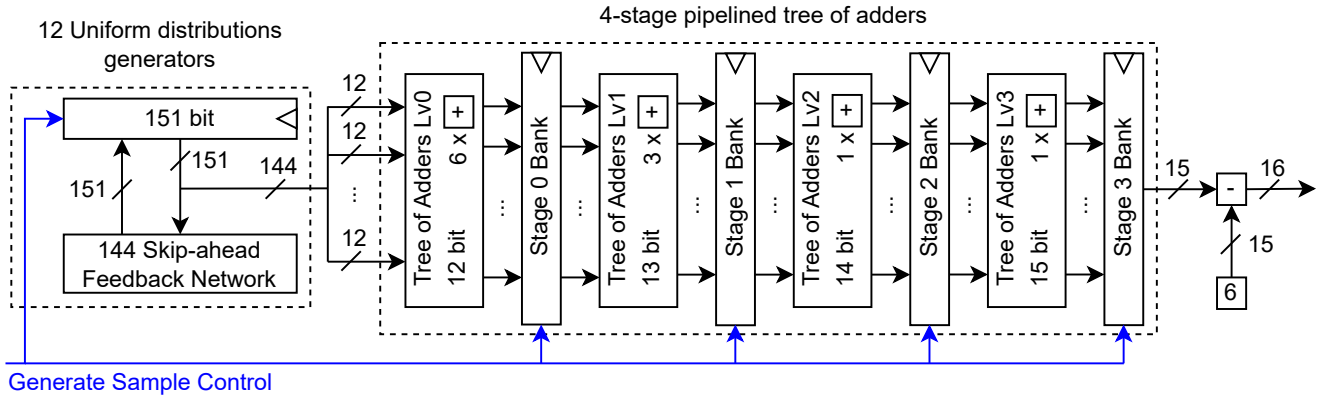


Fig. 4: Pipelined CLT-based GRNG diagram.

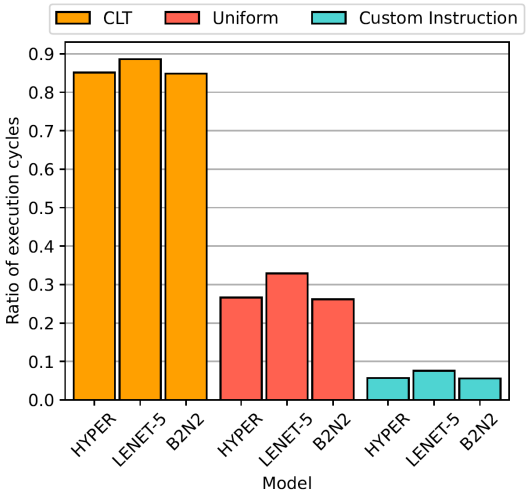


Fig. 5: Inference cycles breakdown of the test models with different weight sampling optimizations.

extension when implemented on the Xilinx ZCU104 FPGA board: 240 Look-Up Tables (LUT) and 320 Flip-Flops (FF). Power measurements where obtained using the power report of Vivado Design suite. This cost includes the GRNG and the modifications of the CPU instruction decoding and the data pipeline. These modifications have no impact on the 100 MHz clock, and only increase the power consumption by 0.65%. Therefore, the reduction in energy consumption is almost identical to the performance gain.

TABLE VI: FPGA resource utilization and power consumption of the baseline RISC-V CPU and the GRNG extension.

	Resource Utilization		**Utilization %**
Type	*Base CPU*	*GRNG Extension*	*Extended CPU*
LUT	2435	240	1.16
Flip-Flop	1922	320	0.49
BRAM	16	0	5.13
DSP	12	0	0.69
	Power consumption (W)		
Type	*Base CPU*	*GRNG Extension*	
Dynamic	0.023	0.004	
Static	0.593	0	

VII. Conclusions

This work studies Bayesian Neural Networks inference running on low power IoT devices. The contributions include publicly available tools to transform TFP floating-point BNN models to C code for integer-only BNN inference on small CPUs. Besides, we analyze different optimization methods for the weight sampling section of the inference algorithm, as they take most of the execution time.

To accelerate the inference, we propose sampling a uniform distribution instead of a Gaussian one. This optimization achieves, on average, $\times 5$ speedup in several representative BNN models. However, It only preserves the accuracy and the uncertainty metrics for small models. To improve these results, we develop a custom RISC-V extension for sampling a Gaussian distribution using a single instruction, targeting a small 32-bit RISC-V CPU. The Gaussian samples are generated using a low cost CLT-based GRNG. The extension accelerates up to $\times 8.10$ times the inference, with similar reductions in the energy consumption, without significant degradation in the accuracy or the uncertainty metrics. We implement and test our design in a Xilinx ZCU104 FPGA. Its hardware cost is only 240 LUTs and 320 Flip-Flops, generating a 0.65% increase in power consumption, without impacting the system clock frequency.

References

[1] A. Alcolea and J. Resano, "Bayesian neural networks to analyze hyperspectral datasets using uncertainty metrics," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–10, 2022.

[2] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Ai and ml accelerator survey and trends," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, 2022, pp. 1–10.

[3] R. Cai, A. Ren, N. Liu, C. Ding, L. Wang, X. Qian, M. Pedram, and Y. Wang, "VIBNN: hardware acceleration of bayesian neural networks," *CoRR*, vol. abs/1802.00822, 2018. [Online]. Available: http://arxiv.org/abs/1802.00822

[4] A. Hiromitsu and H. Masanori, "Bynqnet: Bayesian neural network with quadratic activations for sampling-free uncertainty estimation on fpga," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 1402–1407.

[5] H. Awano and M. Hashimoto, "B2n2: Resource efficient bayesian neural network accelerator using bernoulli sampler on fpga," *Integration*, vol. 89, pp. 1–8, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926022001523

[6] J. S. Malik and A. Hemani, "Gaussian random number generation: A survey on hardware architectures," *ACM Comput. Surv.*, vol. 49, no. 3, nov 2016. [Online]. Available: https://doi.org/10.1145/2980052

[7] J. S. Malik, A. Hemani, J. N. Malik, B. Silmane, and N. D. Gohar, "Revisiting central limit theorem: Accurate gaussian random number generation in vlsi," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 5, pp. 842–855, 2015.

[8] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," 2015.

[9] U. Bhatt, Y. Zhang, J. Antorán, Q. V. Liao, P. Sattigeri, R. Fogliato, G. G. Melançon, R. Krishnan, J. Stanley, O. Tickoo, L. Nachman, R. Chunara, A. Weller, and A. Xiang, "Uncertainty as a form of transparency: Measuring, communicating, and using uncertainty," *CoRR*, vol. abs/2011.07586, 2020. [Online]. Available: https://arxiv.org/abs/2011.07586

[10] S. Pérez Pedrajas, J. Resano Ezcaray, and D. Suárez Gracia, "BNN_RISC-V," 2024. [Online]. Available: https://github.com/Samulix20/BNN_RISC-V

[11] G. Marsaglia, "Xorshift rngs," *Journal of Statistical Software*, vol. 8, no. 14, p. 1–6, 2003. [Online]. Available: https://www.jstatsoft.org/index.php/jss/article/view/v008i14

[12] L. Colavito and D. Silage, "Efficient pga lfsr implementation whitens pseudorandom numbers," in *2009 International Conference on Reconfigurable Computing and FPGAs*, 2009, pp. 308–313.

[13] P. Alfke, "Xilinx. efficient shift registers, lfsr counters, and long pseudorandom sequence generators," July 1996. [Online]. Available: https://docs.xilinx.com/v/u/en-US/xapp052