BnnRV: Hardware and Software Optimizations for Weight Sampling in Bayesian Neural Networks on Edge RISC-V Cores

Samuel Pérez Pedrajas, Javier Resano, and Darío Suárez Gracia, *Member, IEEE*

***Abstract*—Bayesian Neural Networks (BNN) allow prediction uncertainty estimation, making them a more suitable option for safety-critical applications. However, in BNNs, the forward-pass computational cost is significantly higher than in traditional neural networks (NN), due to the overhead generated by weight sampling. This limits their deployment in edge systems. This paper presents an optimization that allows using lower-cost Uniform distribution sampling instead of Gaussian sampling during BNN inference. Building upon this optimization, this paper proposes a lightweight RISC-V instruction set architecture extension that accelerates BNN inference by introducing fixed point arithmetic operations and an efficient Uniform random number generator. The flexibility of RISC-V enables such domain-specific acceleration, narrowing the performance gap between NNs and BNNs for edge machine learning workloads. The proposed software and hardware optimizations achieve an average speedup of 8.93× while reducing energy consumption per forward pass by 87.12%, increasing image/J efficiency by 8.19×. They have been designed to maintain accuracy, calibration, and uncertainty quality, while optimizing execution efficiency. This has been verified with an extensive validation process that considers relevant model architectures. Additionally, our results highlight that weight sampling is no longer the BNN inference performance bottleneck, shifting the primary limiting factor to control overhead.**

***Index Terms*—Bayesian neural networks, RISC-V, edge computing, tiny machine learning.**

I. Introduction

NEURAL Networks (NN) have revolutionized a multitude of tasks. For example, in image classification, NNs have reached previously unseen accuracy levels on complex datasets like ImageNet [1]. Recently, generative AI has further extended NNs, enabling advances in natural language processing and image generation. Reaching this level has required deep architectures that have significantly increased their number of parameters and computational requirements [2]. As NNs continue to grow, so do their energy demands, which questions the sustainability of these models. Moreover, it also reduces the transparency of the models. In response, different organizations, such as the European Union [3], or UNESCO [4], have demanded improving transparency and trust in AI systems while encouraging the development of more energy-efficient AI architectures. Edge AI is one of the technologies that can help to achieve these goals. In fact, in the recent European Innovation Council Tech Report [5], edge AI was identified as one of the emerging technologies that could change our future.

TinyML refers to small and efficient machine learning (ML) models deployed on resource-constrained, low-power devices. Traditional Internet of Things (IoT) systems rely on sensor devices deployed on the edge to collect data, which is then sent to large data centers for analysis. However, IoT systems face challenges such as latency, high energy consumption, both in network transfers and data centers, and data privacy concerns. The TinyML approach enables processing data directly on edge systems, taking advantage of underutilized hardware resources, and increasing overall energy efficiency. This, in turn, improves data privacy and reduces network communication and associated latency [6].

Edge devices are typically low-cost, which facilitates their deployment, but this affordability comes at the cost of limited computational and storage resources, restricting their capability of executing large ML models such as NNs.

Significant software efforts have focused on reducing the size and computational demands of NNs leading to efficient models that maintain high performance while being deployable on resource-constrained devices such as smartphones [7], [8]. Pruning and quantization techniques have been shown to be able to significantly reduce model sizes with very little performance degradation [9], [10].

Hardware development efforts are also being made to accelerate NNs, with architectures specialized to perform matrix multiplications, such as tensor processing units (TPU) [11]. Main smartphone chip manufacturers, like Apple [12], Huawei [13] or Qualcomm [14] are following this approach, adding hardware ML accelerators to their chips.

Received 20 February 2025; revised 2 August 2025 and 8 October 2025; accepted 19 October 2025. Date of publication 24 October 2025; date of current version 31 March 2026. This work was supported in part by the Grant PID2022-136454NB-C22, in part by AEI/10.13039/501100011033, "ERDF A way of making Europe," in part by the PDC2023-145851-I00 grant, funded by AEI/10.13039/501100011033, "European Union NextGenerationEU," and in part by the Government of Aragon (T58_23R research group). The review of this article was arranged by Associate Editor Chenchen Liu. *(Corresponding author: Samuel Pérez Pedrajas.)*

The authors are with the Department of Computer Sciences and Systems Engineering (DIIS), Aragon Institute of Engineering Research (I3A), University of Zaragoza, HiPEAC European NoE, 50018 Zaragoza, Spain (e-mail: samuel.perez@unizar.es).

Digital Object Identifier 10.1109/TCASAI.2025.3625517

NN hardware acceleration can also be used to overcome the performance limitation of edge devices. This acceleration can be implemented using a microcontrol unit (MCU) coupled with an FPGA or an ASIC, such as an Edge TPU [15]. Adding hardware acceleration can significantly boost NN performance, but it also increases costs and often reduces program flexibility, which is not desirable in the IoT market. The RISC-V open instruction set architecture (ISA) offers another way to address the challenge of TinyML. The RISC-V ISA can be extended with domain-specific instructions, allowing for optimized performance on ML tasks while retaining program flexibility.

When deploying TinyML models on edge devices, battery life is a central issue, as these devices often operate in environments where frequent recharging or battery replacement is impractical. Although TinyML aims for small model size and low latency, there is still a trade-off between computational intensity and energy consumption. Therefore, practical TinyML deployment requires optimization of model architecture and hardware selection to maximize battery life.

Performance and energy efficiency are important, but TinyML applications also demand trustworthy ML models, especially in safety-critical scenarios such as self-driving cars, health monitoring, or threat detection. Modern NNs have been found to be poorly calibrated, often displaying overconfidence in incorrect predictions [16]. ML models that are able to determine the prediction uncertainty, such as Bayesian Neural Networks (BNN), are more suited for this kind of applications.

However, deploying BNNs in TinyML environments poses specific challenges. Due to their probabilistic nature, BNNs require greater computational and memory resources, which can conflict with the constraints of edge devices. Despite these limitations, the use of BNNs is becoming increasingly relevant in edge computing, where reliability and uncertainty estimation are critical. Unlike deterministic models, BNNs incorporate probabilistic modeling, enabling the quantification of predictive uncertainty [17], a key feature in safety critical environments. This allows edge devices to measure the confidence of their inferences in real time, adapt to ambiguous or out of distribution inputs and make more robust decisions.

Deploying BNNs on edge devices enables trustworthy inference, minimizing dependence on cloud connectivity while enhancing privacy and responsiveness. Furthermore, in real-world IoT environments where data is often incomplete, noisy, or corrupted, BNNs have shown greater robustness compared to traditional neural networks [18].

Unlike traditional NNs with fixed values as weights, BNNs represent weights as probability distributions, typically Gaussian, and, during inference, produce an output distribution by sampling these distributions. Therefore, BNNs enable us to quantify the uncertainty at the end of the process [19]. One key drawback is that a single forward pass in a BNN takes approximately 12.16× more time than in a conventional NN mainly due to the cost of sampling. In addition, the probabilistic nature requires multiple forward passes per inference, significantly increasing computational overhead. The exact number of passes depends on the application. Previous studies have explored how many samples are needed for the convergence of uncertainty metrics, with some reporting up to 100 forward passes [20].

The contributions of this paper are the following.

- An open-source toolchain, called BnnRV, to transform BayesianTorch [21] trained BNN models into integer-only inference C code that can be executed in small single-core MCUs [22].
- The development of a novel weight sampling optimization technique and an extensive evaluation of its impact considering key BNN performance-related metrics across relevant classification model architectures. With regard to performance, this optimization obtains 4.94× average speedup.
- An open-source, low cost RISC-V ISA extension designed for BNN inference acceleration that achieves an average 8.93× speedup and 87.12% energy reduction compared to the baseline BnnRV implementation on a 32 bit RISC-V core [23], effectively narrowing the performance gap between NN and BNN inference from 12.16× to just 1.37× slower.

II. Related Work

The RISC-V for IoT ecosystem is rapidly expanding with open-source processor projects e.g.: the PicoRV32 [24] and MicroRV32 [25] are both general-purpose, configurable low-cost cores.

Other works, which target TinyML applications such as edge deployment of NNs, have proposed their own low-cost processors coupled to RISC-V ISA extensions. Mr. Wolf is a parallel ultra low power (PULP) cluster of extended RISC-V cores that can deliver up to 274 MOp/s/mW [26]. As low-cost MCUs often lack support for floating-point arithmetic, these SoCs prefer quantized fixed-point integer-only NN inference. Therefore, the authors incorporate fixed point arithmetic, single instruction multiple data (SIMD), and DSP-like instructions, such as zero overhead hardware loops on top of the ISA in the cores of the cluster [27]. XpulpNN proposes a set of RISC-V ISA extensions that aim to accelerate the inference of highly quantized NNs implemented in a PULP RISC-V architecture [28]. They suggest the use of SIMD techniques to exploit the data parallelism and smaller data type sizes of quantized NNs.

Beyond TinyML applications, recent research has explored the deployment of foundational models, such as large language models, in custom-extended RISC-V clusters, achieving significant performance gains [29].

Given that BNN inference involves greater computational complexity compared to classic NNs, previous efforts have predominantly explored FPGA-based discrete accelerators. Within this domain, the VIBNN accelerator focuses on optimizing the Gaussian RNG required for the inference algorithm [30]. The final efficiency peaks at 52694.8 Images/J without accuracy degradation. The design proposed two different Gaussian RNGs, one based on the central limit theorem (CLT) relation with the binomial distribution, and another based on the Wallace method.

However, recent work has shown that the precision of the Gaussian RNG required for inference is relatively low [31]. In this work, the authors propose utilizing a 32-entry, 6-bit LUT to create a Gaussian RNG based on inverse transform sampling.

The B2N2 accelerator, following the principle of low precision requirement, introduced the use of Bernoulli RNGs instead of Gaussian RNGs, applying a transformation to the weights after training [20]. As these generators have low hardware cost, the energy efficiency is improved by 57.5%, compared to the VIBNN accelerator, with a minimal accuracy drop.

One work tried to solve the problem of needing multiple forward passes by using quadratic activation functions instead of standard nonlinear ones [32]. This worked well for small models, but later research by the same authors found that it did not perform as well on larger models [20].

Another work proposed approach involves storing precomputed coefficients [33]. This method modifies the inference process by first calculating coefficient matrices for a layer using the input, along with the mean and variance of the weights, this step is done only once. After that, the multiple forward passes become less computationally intensive, using an uncertainty matrix as input. However, the downside is that this technique more than doubles the model's memory usage, which may make it unsuitable for edge deployment. Nevertheless, we see this method as complementary to our own and believe that it could be used alongside it, even though we have not tested this combination.

Other work explored using stochastic computing to build a discrete accelerator for BNNs [34]. In this computing paradigm, data is represented by random bitstreams, where the value is encoded as the proportion of 1s in the stream. This allows for the use of simple 1-bit logical operations instead of traditional arithmetic. Taking advantage of the simplicity of these operations and the probabilistic nature of stochastic computing, the authors reported significant efficiency improvements in BNN inference. However, their work only showed a proof of concept on a small model with just four fully connected layers. It would benefit from testing on larger models and evaluating not just accuracy, but also the quality of uncertainty estimation.

In the context of ISA extensions for in-core acceleration, previous research incorporated a CLT-based GRNG into a low-cost RISC-V processor to optimize BNN inference [35]. This work also proposes using Uniform distributions to optimize the weight sampling process but reported significant accuracy losses in models with more than 3 linear layers. Our work overcomes these limitations by developing sampling techniques that can be applied efficiently to models of any size and demonstrates it with an extensive validation process. For this purpose, we take a different approach by adding a more hardware efficient Uniform RNG alongside fixed-point arithmetic support.

On the software side, specialized libraries support BNNs, including the Python ML frameworks TensorFlow [36] and PyTorch [37]. TensorFlow-Probability [38] extends TensorFlow, while BayesianTorch [21] integrates with PyTorch. These libraries are primarily optimized for training on high-performance GPU systems rather than for embedded deployment. Recent advances include the development of post-training quantization methods for BNNs and techniques such as a BNN-specific adaptation of Batch Norm Folding [39]. However, current frameworks for embedded NN deployment, such as TensorFlow Lite, PyTorch Edge, and Micro Tensor Virtual Machine (TVM) [40], do not yet offer a streamlined solution to deploy BNNs on embedded hardware.

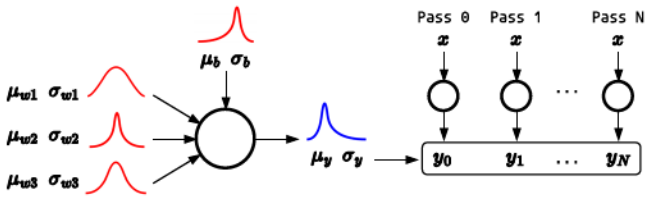

Fig. 1. Diagram of a Bayesian neuron. The weights and bias are modeled as probability distributions defined by their means and standard deviations. Multiple forward passes are performed using samples from these distributions to produce an output distribution, capturing uncertainty.

Unlike previous related work, our work focuses on optimizing BNN inference rather than traditional NNs, and we propose a novel software toolchain and a tiny RISC-V ISA extension that incorporates a fixed-point multiply accumulate (MAC) instruction and a Uniform RNG instruction. This approach targets, both from a hardware and software perspective, the key aspects of BNN optimization, allowing efficient execution of uncertainty-aware ML models in low-power systems. After these optimizations, the remaining operations in BNN inference are almost the same as those of conventional NNs, allowing the application of standard NN optimizations orthogonally.

III. Bayesian Neural Networks Background and Evaluation Metrics

BNNs extend traditional NNs by representing both weights and predictions as probability distributions, allowing them to capture uncertainty in their outputs. Instead of producing a single deterministic result, BNNs require multiple forward passes to sample from these distributions and generate the output distribution. Fig. 1 shows a diagram of a Bayesian neuron.

To estimate the posterior distribution $p(w|D)$ of a set of weights w given an observed training dataset $D = \{x, y\}$ BNNs rely on the Bayes theorem. Calculating this posterior distribution for complex models like NNs is unfeasible. During training, variational inference is used to approximate the posterior distribution using a simpler parametrized distribution [19].

TensorFlow Probability and BayesianTorch utilize Gaussian distributions as the approximating distributions due to their favorable mathematical properties. With Gaussian distributions, the network parameters consist of a set of means and standard deviations $\phi = \{\mu, \sigma\}$, serving as the model weights during both training and inference. As a result, for the same architecture, BNNs require twice the number of parameters than traditional NNs.

Once a model is trained, Equation 1 estimates the probability of a new observation x^* belonging to a class y^*.

$$p(y^*|x^*, D) = \int p(y|x^*, w)p(w|D)dw \tag{1}$$

Monte Carlo (MC) sampling can be used to approximate the integral of Equation 1, requiring multiple stochastic forward passes where the Gaussian distributions that form the approximate posterior are sampled.

A. Model Calibration

Model calibration refers to the process of aligning the confidence of a model's predictions with the actual probability of those predictions being correct. The Expected Calibration Error (ECE ↓) can measure the difference between the model confidence and accuracy probability [41].

The Reliability Error (RE ↓) is derived from the reliability plot described by Alcolea et al. [42]. RE represents the average error between expected and predicted probabilities. Compared with ECE, RE considers both the predicted class confidence and the whole vector of class probabilities of Equation 1.

The combination of these two metrics can determine whether the model outputs can be reliably interpreted as probabilities, which is important when they are used to make decisions.

B. Uncertainty Quantification

The predictive uncertainty $\mathbb{H}(y|x, D)$ of a prediction in a classification setting can be quantified using Equation 2 [43]. Let K be the number of classes and T the number of MC samples a_t.

$$\mathbb{H}(y|x, D) = -\sum_{k=1}^{K}\left[\left(\frac{1}{T}\sum_{t=1}^{T} a_t\right)\log\left(\frac{1}{T}\sum_{t=1}^{T} a_t\right)\right] \quad (2)$$

Predictive uncertainty can be further divided into aleatoric uncertainty, captured by the expected entropy and related to the quality of the training dataset, and epistemic uncertainty, captured by the mutual information and related to the quality of model training and architecture [44].

C. Uncertainty Evaluation

In the case of BNNs, the model predictive uncertainty must also be calibrated. A well-calibrated uncertainty should be high when a prediction has a high probability of being inaccurate. The expected uncertainty calibration error (UCE ↓) measures the difference between the model predictive uncertainty and the error probability [45].

Model uncertainty quality can also be evaluated using conditional probabilities such as $p(accurate|certain)$ (PAC ↑) and $p(uncertain|innacurate)$ (PUI ↑) [44]. A prediction is considered certain or uncertain given an uncertainty threshold u_{th}. These probabilities can be used to evaluate how well the predictive uncertainty can be used to estimate the accuracy, and therefore consider trusting a prediction.

IV. Methodology

To ensure the correctness of the proposed optimizations, the proposed toolchain includes a set of diverse and relevant test cases, whose results have been compared with the original models of the BayesianTorch Python library [21].

The validation process relies on the calibration and uncertainty evaluation metrics detailed in Section III. For testing, models were trained using the MOPED framework in BayesianTorch [46], if no trained versions were available. The different model architectures used in the testing process are listed below.

Hyperspectral Pixel Classification Models A set of small models for hyperspectral pixel classification. These models were developed for on-board remote sensing and are therefore a relevant case of edge computing. All of them have a simple architecture with three fully-connected linear layers. Each model was trained to classify pixels of one of the following hyperspectral images: Botswana (BO), Indian Pines (IP), Kennedy Space Center (KSC), Pavia University (PU), and Salinas Valley (SV). The trained versions of the models were provided by Alcolea et al. [1], who verified that they were well calibrated [42].

Bayesian-LeNet-5: A convolutional BNN model with the same architecture as the LeNet-5 model [47] trained for the classification of the CIFAR-10 image dataset [48].

B2N2: The performance evaluation model introduced by Awano et al. along with the B2N2 accelerator was also trained for image classification on the CIFAR-10 dataset [20]. It follows a VGG-like architecture [49].

Bayesian-TinyResNet: A residual BNN model with the same architecture as the ResNet model described in MLPerf™ Tiny benchmark for the classification of the CIFAR-10 image dataset [50]. This benchmark is a set of different relevant problem models that can be used for performance evaluation in TinyML environments.

To the best of our knowledge, this work presents the most comprehensive evaluation of models used in BNN based accelerators. The analysis spans a wide range of architectures, including compact models designed for ultra low power devices, the ones used for hyperspectral classification, large convolutional networks such as the B2N2 model, and residual networks like TinyResNet, adopted by MLPerf Tiny as the reference model for embedded image classification tasks. This work specifically targets TinyML and edge devices, where resource constraints limit the deployment of significantly larger models, which are not suited for such environments.

For performance evaluation, this work uses an open-source, 32-bit, 5-stage, in-order RISC-V processor with integer multiplication support [23]. The processor is very simple and suitable for low power systems, achieving a similar score in the Embench™ [51] matmul benchmark to reference open-source cores such as the RI5CY core [27].

This processor was also the platform extended with new instructions. The RISC-V GNU toolchain 13.2.0 with `-O3` enabled was used for cross-compilation. To achieve cycle-accurate measurements, the BnnRV inference code was executed in a simulated environment using Verilator 5.024. For cost and energy efficiency evaluation, the different processor designs were

[1] https://github.com/universidad-zaragoza/BNN_for_hyperspectral_datasets_analysis

implemented in a Zynq UltraScale+ ZCU104 evaluation board FPGA using Xilinx Vivado 2023.2.

The FPGA validation device used in this work is not intended for actual edge deployment, it is only used to estimate hardware cost and power consumption, but was chosen to maintain consistency with prior studies, which also used it for evaluation, and ensure a fair comparison. Since the proposed implementation in this work uses only about 2% of the FPGA resources, it can be implemented in a much smaller edge-oriented FPGA, which would reduce cost and power consumption.

V. The BnnRV Toolchain

BnnRV is a toolchain designed to generate optimized source C code for the inference of BNN models trained using BayesianTorch. The structure of the generated C code is such that eases the automatic optimization by the RISC-V GNU C compiler; e.g., function inlining or loop unrolling. By translating trained models into C, BnnRV enables the deployment of uncertainty-aware NNs on resource-constrained devices.

Like the aforementioned embedded NN frameworks, BnnRV employs multiple optimization techniques. For example, inference runs with integer fixed-point precision values. NN models, including BNNs, are generally resilient to changes in operand precision due to their redundant and distributed computations, a key reason why quantization is often effective.

In addition, BnnRV minimizes memory usage by reusing two buffers for the input and output of the model layers, or three buffers when residual blocks require so. Besides, to reduce loop control instructions overhead, the toolchain fuses activation function layers with their preceding layers. Additionally, to eliminate the computational cost of Batch Normalization layers, BnnRV incorporates a BNN-specific adaptation of Batch Norm Folding optimization [39].

A forward pass of a BNN requires sampling the weight distributions learned during training. These distributions are Gaussian and need to be generated using a Gaussian RNG algorithm. BnnRV relies on a CLT-based algorithm that accumulates 12 Uniform distribution samples. For Uniform random number generation, BnnRV utilizes the 32-bit Xorshift [52].

Fig. 2 shows that the Gaussian weight sampling algorithm takes up the majority of execution cycles, making it a primary target for optimization. Batch inference can reduce the cost of weight sampling because batch elements can share weights among them. However, this approach raises the inference memory requirements and may not be allowed in real-time IoT applications due to limited resources or real-time restrictions. Since BnnRV focuses on BNN inference in resource-constrained environments, reducing the sampling delay using batching is not a suitable option.

A. Using Uniform Distributions for Weight Sampling

Gaussian sampling remains computationally expensive, even when using a simple algorithm, as shown in Fig. 2. To address this issue, this paper proposes an optimization that replaces Gaussian distributions with Uniform distributions for inference, which greatly reduces weight sampling computational

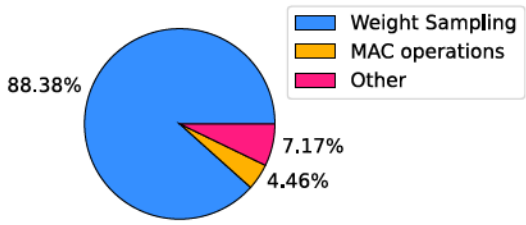

Fig. 2. Ratio of the average execution cycles for all test models using Gaussian weight sampling, categorized into Weight Sampling, MAC operations and other.

complexity. This optimization is designed to be implemented without changing the model training process.

The core MAC operation of a BNN neuron is shown in Equation 3, where y denotes the output, w and x represent the input vectors of size N, and b is the bias term. Since these vectors are typically large, the CLT can be applied, allowing the output y to be approximated as a Gaussian distribution, $y \sim \mathcal{N}(\mu_y, \sigma_y)$.

$$y = b + \sum_{i}^{N} w_i x_i \tag{3}$$

Since a Gaussian distribution is fully described by its mean and variance, the output of the MAC operation can be described accordingly. Equation 4 shows how μ_y and σ_y^2 are obtained.

$$\begin{aligned} \mu_y &= \mu_b + \sum_{i}^{N} \mu_{w_i} \mu_{x_i} \\ \sigma_y^2 &= \sigma_b^2 + \sum_{i}^{N} \sigma_{x_i}^2 \sigma_{w_i}^2 + \sigma_{w_i}^2 \mu_{x_i}^2 + \sigma_{x_i}^2 \mu_{w_i}^2 \end{aligned} \tag{4}$$

The CLT ensures that the output distribution of the MAC operation converges to a Gaussian regardless of the distributions of the inputs. Therefore, the behavior of the output is determined by the means and variances of the MAC operands as was shown in Equation 4. To preserve the behavior of the BNN model, it is sufficient to maintain the same expected value and variance for the output. This allows the use of alternative distributions for the weights, such as Uniform, in place of Gaussian. For a weight originally sampled from a Gaussian distribution $w \sim \mathcal{N}(\mu_w, \sigma_w)$, an equivalent weight can be defined with a Uniform distribution using the transformation $w' \sim b\,\mathcal{U}(0,1) + a$. This weight is also parameterized by only two values, a and b, meaning that the memory footprint of the model remains the same. The parameters a and b can be computed from μ_w and σ_w, as shown in Equation 5. Importantly, this transformation is applied only once after training, without requiring any modifications to the training process.

$$\begin{cases} \mu_w = b/2 + a \\ \sigma_w^2 = b^2/12 \end{cases} \quad \begin{cases} a = \mu_w - b/2 \\ b = \sigma_w \sqrt{12} \end{cases} \tag{5}$$

B. Toolchain Validation

To evaluate and ensure that both accuracy and uncertainty quality are preserved across models of varying size and architecture, a precision analysis was performed during validation. This involves varying the fixed-point bit width of inputs,

TABLE I
VALIDATION METRICS OF THE TEST MODELS OBTAINED USING BNNRV WITH GAUSSIAN AND UNIFORM GENERATION

	Accuracy % ↑			ECE % ↓			UCE % ↓			RE % ↓		
	BayesianTorch	BnnRV-Gaussian	BnnRV-Uniform	BayesianTorch	BnnRV-Gaussian	BnnRV-Uniform	BayesianTorch	BnnRV-Gaussian	BnnRV-Uniform	BayesianTorch	BnnRV-Gaussian	BnnRV-Uniform
Model	Val	Diff	Diff	Val	Diff	Diff	Val	Diff	Diff	Val	Diff	Diff
BO	90.52	-0.06	0.00	0.88	0.68	0.21	2.13	0.35	0.22	2.73	1.39	0.46
IP	81.44	0.12	0.18	0.94	-0.14	0.13	4.34	-0.41	-0.33	2.40	-0.11	0.11
KSC	92.63	-0.15	-0.12	3.60	-0.17	0.01	5.79	-0.32	-0.23	6.49	-0.61	-0.33
PU	90.11	0.02	0.03	2.68	-0.06	-0.05	2.13	-0.10	-0.10	5.75	-0.08	-0.11
SV	92.58	0.10	0.05	0.75	-0.05	-0.11	2.18	-0.09	-0.10	2.26	0.45	-0.14
LENET	62.61	-0.32	-0.38	2.66	-1.28	-1.16	4.09	1.70	1.35	2.62	-0.84	-0.75
B2N2	75.77	0.35	0.17	1.69	-1.01	-1.07	2.72	2.12	1.86	2.13	-0.42	-0.54
RESNET	81.01	-1.26	-1.34	1.61	-0.73	-0.61	2.24	0.81	0.71	2.23	-0.88	-0.74
Average		-0.15	-0.18		-0.34	-0.33		0.51	0.42		-0.14	-0.25
Std. Dev.		0.49	0.50		0.63	0.54		0.96	0.81		0.76	0.42

weights, and activations independently and measuring their impact on classification accuracy as well as uncertainty metrics. By comparing these results against full precision baselines, configurations that maintain acceptable performance can be identified. Evaluation across multiple model types, including fully connected, convolutional, and residual architectures, shows that this approach preserves accuracy and uncertainty quality with minimal degradation using ten bits for fixed-point fractional part.

In the literature, more aggressive quantization techniques with bigger bit width reductions have been proposed for traditional NNs, but require additional training steps and are not supported by standard BNN design environments. In contrast, this work uses the original PyTorch models as the starting point and does not require any additional iteration in the training process. In any case, the number of bits used for fixed-point arithmetic is a parameter that can be easily adjusted in the BnnRV environment.

Table I presents the BnnRV results using both Gaussian and Uniform distributions for all test models. The results show that the BnnRV optimized models obtain an average accuracy almost identical to the original BayesianTorch models, with a maximum accuracy loss of 0.15% and 0.18% for BnnRV-Gaussian and BnnRV-Uniform, respectively. Regarding the other metrics, the results show small deviations that do not result in performance degradation. There is a decrease in ECE of 0.34% for BnnRV-Gaussian and 0.33% for BnnRV-Uniform. An increase in UCE of 0.51% for BnnRV-Gaussian and 0.42% for BnnRV-Uniform. And a decrease in RE of 0.14% and 0.25% for BnnRV-Gaussian and BnnRV-Uniform respectively. Therefore, accuracy, output calibration, and uncertainty metrics are very similar to those of the original models.

The worst drop in accuracy is 1.34% for the RESNET model using the Uniform distribution, and the worst case of an increase in UCE is 2.12% for the B2N2 model using the Gaussian distribution. The best improvement cases are a 1.28% drop in ECE for the LENET model using the Gaussian distribution and a 0.88% drop in RE for the RESNET model using the Gaussian distribution. These variations can be attributed to the probabilistic nature of the models. Additionally, the loss of representation precision when transitioning from floating-point to fixed-point arithmetic may also contribute to these small differences.

To evaluate the differences in the conditional probability metrics, PAC and PUI, the mean squared error (MSE) was calculated between BayesianTorch and BnnRV curves across 50 uncertainty thresholds. The evaluation resulted in a worst-case error of 94.1×10^{-5} for the PAC curve and 8.11×10^{-5} for the PUI. For clarity, these worst-case scenarios are illustrated in Fig. 3, which demonstrate the similarity between both curves.

Fig. 4 compares the confidence and uncertainty output distributions with those of the baseline, separated into matching and differing predictions. The plots demonstrate a high degree of similarity between the distributions. This implies that, after the optimizations performed, the usefulness of the uncertainty metrics to estimate the accuracy of a prediction is similar to that of the original models.

While previous work suggested that this optimization was limited to small models [35], the results presented in this section demonstrate otherwise. With a more extensive validation across all metrics and a new software implementation of the optimization, this work shows that it can be applied to BNN models regardless of their size.

VI. EXTENDING RISC-V TO FURTHER OPTIMIZE BNNRV

As mentioned previously, the BnnRV inference library utilizes fixed-point arithmetic, which is implemented entirely with integer instructions. The code for BNN inference, incorporating the previously discussed Uniform weight optimization, relies on a Uniform RNG algorithm and two fixed-point MAC operations: the first for weight generation and the second for the standard weight accumulation used in NNs. A fixed-point MAC operation involves a bit shift to adjust the scale after the

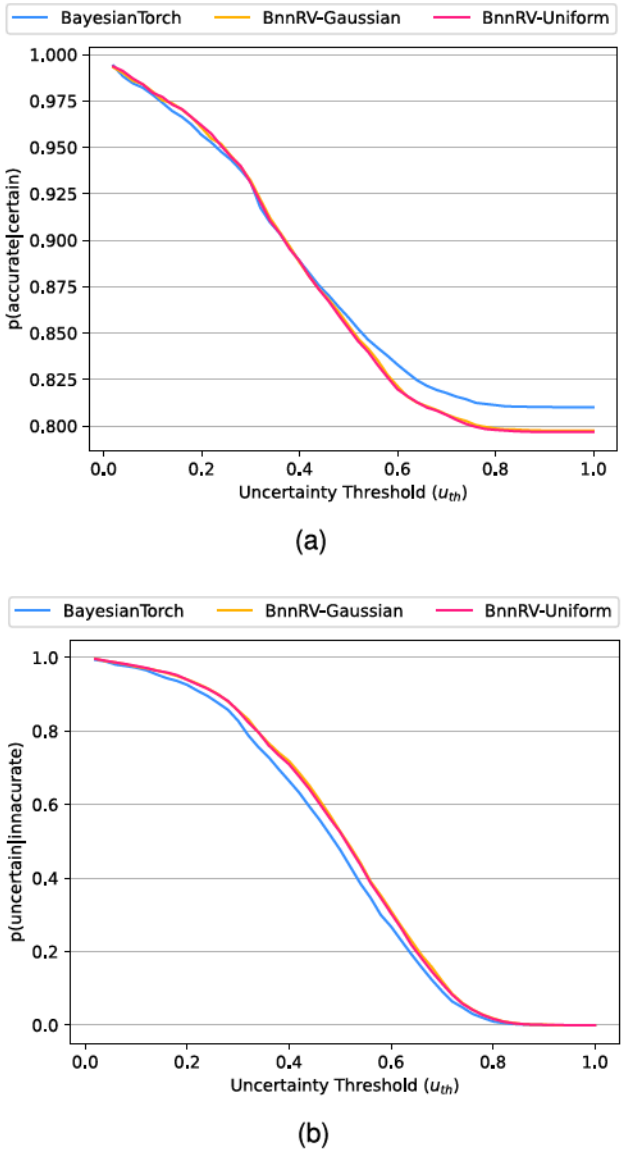

Fig. 3. Worst case models PAC (a) and PUI (b) curves comparing BayesianTorch, BnnRV-Gaussian, and BnnRV-Uniform.

multiplication, resulting in a total of three instructions per MAC operation.

This paper proposes enhancing the performance of BNN inference with two new key instructions, a fixed-point MAC, and a Uniform RNG. Table II details the new instructions. In addition, a complementary instruction for random seed configuration was included for completeness.

Using this extension, the critical computation of BNN inference only requires executing three assembly instructions, which will require only three cycles in our RISC-V core.

The fixed-point scale represents the number of bits assigned to the fractional part of a number and can be stored using 5 bits for 32-bit precision. The `fx.madd` instruction uses the R4 RISC-V encoding. This encoding provides 5 control bits divided in two fields, `funct2` and `funct3`. The 5-bit size allows for encoding the scale value, used to define the number

TABLE II
CUSTOM RISC-V INSTRUCTION PROPOSAL DESCRIPTIONS

Instruction	Description
`fxg.unif rd, I`	`rd = urng() » I` Generates a Uniform random sample and shifts it by I bits
`fxg.seed ra`	`urng.seed(ra)` Sets the seed of the generator using the value of a register
`fx.madd rd, ra, rb, rc, I`	`rd = ra + (rb * rc) » I` Performs a fixed point MAC operation with I scale

of bits assigned to the fractional part of a number, within those fields as an immediate, within the 32 bits of the instruction format. Using immediate values means that the code needs to be recompiled every time the fixed point scales change, which should not often occur after the BNN model is deployed.

Implementing `fxgen.unif` requires adding a new uniform RNG functional unit. The implementation of the `fx.madd` instructions requires extending the base RISC-V processor with a third read port to the integer register file and a dedicated fixed-point MAC functional unit. This also requires extending the instruction decoding logic and adding the necessary connection buses for the new functional unit. For ML-targeted systems, adding a MAC operation is almost mandatory, and including a third register file read port for it is common practice, as done in the RISC-V vector extension and in one of the most popular embedded RISC-V ML cores [27]. The FPGA implementation confirms that the added hardware introduces minimal overhead in both area and power.

These requirements can be easily integrated into any simple RISC-V core. This work proposes two possible implementations of the dedicated functional units. First, a version focused on simplicity and modularity, where both functional units are added independently at the execution stage. These units are presented in Figs. 5 and 6. The clock and reset signals are shown in purple, control signals generated at the decode stage are colored in blue, data operands are shown in yellow and functional unit outputs in green. The `funct2` and `funct3` values are taken directly from the instruction fields and concatenated as a 5 bit immediate used as an operand for bit shifting in fixed-point arithmetic.

To implement the Uniform RNG functional unit, a look-ahead linear feedback shift register (LFSR) was used. The LFSR consists of a register and a feedback network that uses XOR gates to implement a generating polynomial, producing one random bit per cycle. However, generating multiple bits with low correlation requires a more complex method. A look-ahead LFSR applies the generating polynomial multiple times per cycle using a more complex feedback network [53]. To generate 32-bit samples, this work utilizes a 39-bit LFSR with a 32-step look-ahead mechanism and a shifter to set the scale of the sample. The fixed-point arithmetic unit uses a discrete 32-bit multiplication hardware, a 32-bit adder, and a shifter.

This work also proposes a more optimized implementation, which combines both functional units and shares the shifter hardware. In addition, instead of using a discrete multiplier, it

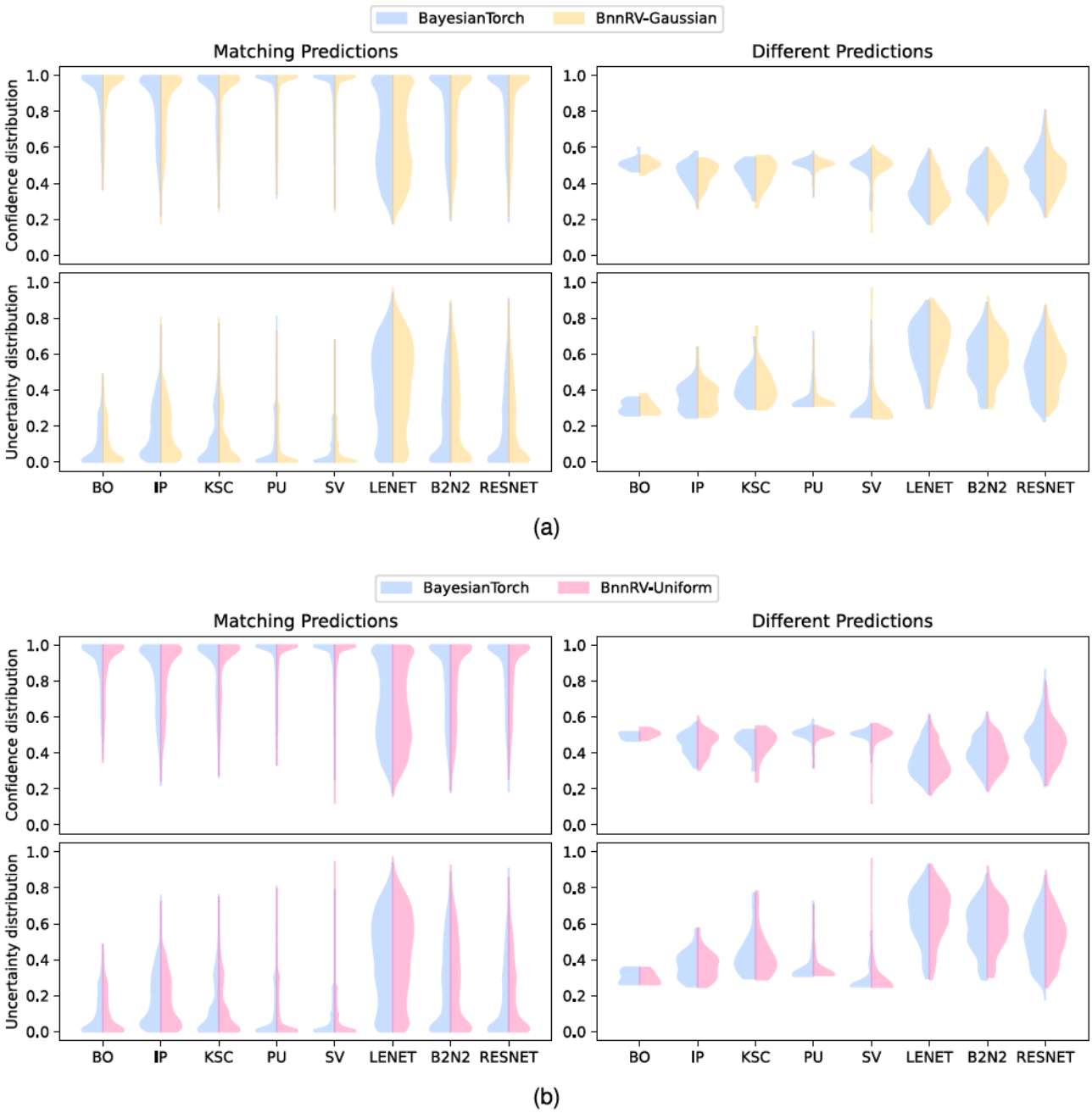

Fig. 4. Comparison of confidence and uncertainty distributions between BayesianTorch and BnnRV with Gaussian (a) and Uniform (b) sampling methods. The distributions are divided into predictions that match and differ from the baseline across various models. BnnRV-Gaussian.

uses the multiplier hardware already present in the base RISC-V core, reducing the area requirements. Fig. 7 shows the diagram of this implementation. In the pipeline, the already existing multiplier's output is colored in red and acts as an input of this new functional unit.

A. Performance and Cost Evaluation

Uncertainty quantification requires multiple passes of an input through a BNN. The number of passes performed is a design decision. This section presents efficiency results for a single input image forward pass. Due to the nature of the Bayesian process, the results scale linearly with the number of passes chosen to achieve the desired quality of the uncertainty metrics.

This section details the performance evaluation of software-only optimizations (S-OPT) and the combination of software and hardware optimizations (SH-OPT) compared to a baseline. The baseline results are obtained running experiments on the testbench processor [23] without any proposed optimizations or extensions. S-OPT results are also obtained by executing the experiments on the unmodified processor, incorporating only software modifications.

Fig. 8 plots the relationship between model size and execution time in terms of required operations and cycles per single

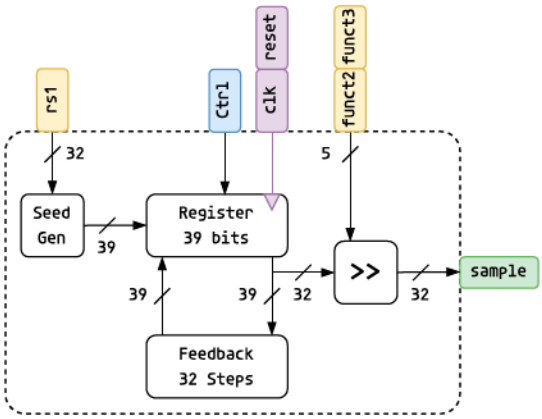

Fig. 5. Uniform RNG functional unit. Functional unit diagrams of the modularity focused implementation of the proposed RISC-V extension.

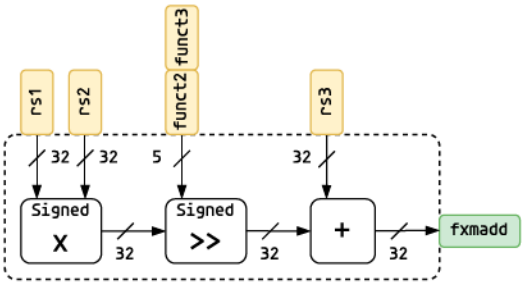

Fig. 6. Fixed point MAC functional unit. Functional unit diagrams of the modularity focused implementation of the proposed RISC-V extension.

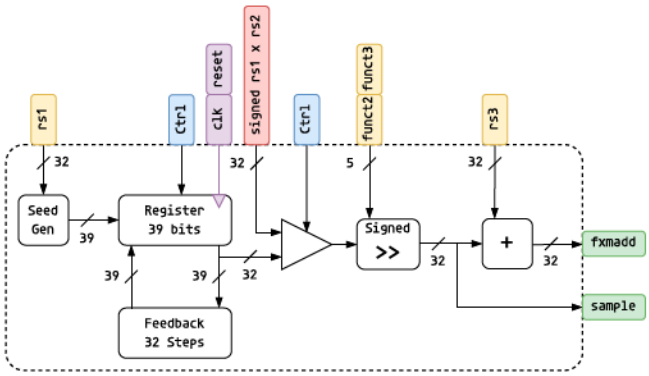

Fig. 7. Functional unit diagram of the optimized implementation of the proposed RISC-V extension.

classified image. This work defines a Bayesian operation as the combination of generating a weight sample and performing a MAC using that sample. The HYPER category groups all hyperspectral pixel classification models. S-OPT achieves an average speedup of 4.96×, while SH-OPT reaches 9.12×. The results also indicate that the remaining models obtain similar performance gains independently of their size.

Fig. 9 presents an execution profiling of all models using the evaluated optimizations. The execution cycles are divided into three categories: weight sampling, MAC operations, and other

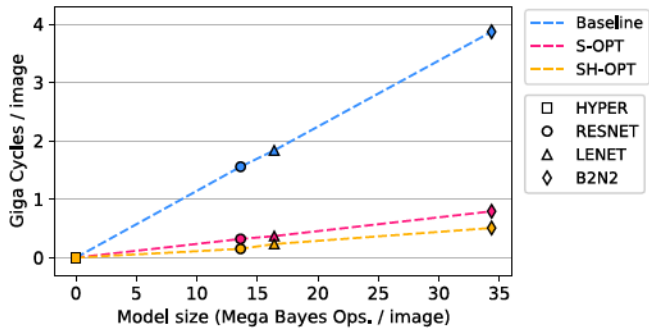

Fig. 8. Relation between model size in Bayesian operations and the number of cycles required to process an image. The different model architectures are represented by shapes and the execution environment by colors.

instructions. The height of the bars reflects the relative speedups achieved by the different configurations.

On average, sampling a Uniform distribution using the software method proposed in this paper achieves a speedup of 4.94×, while utilizing the proposed RISC-V instructions results in a 8.93× speedup. The variation in speedup across models can be attributed to differences in layer architectures. Models with a higher proportion of convolutional layers tend to have greater overhead due to the increased looping involved. Larger layers may also prevent the compiler from unrolling certain loop bodies, introducing additional control overhead and impacting overall performance. Custom instructions reduce the size of the loop body, which in turn enables the compiler to unroll loops more effectively. These factors contribute to the observed differences in speedups, with some models benefiting more from simpler layer structures and better compiler optimizations. An example of this can be seen in the RESNET model, where the use of custom instructions allowed the unrolling of the inner loops of convolution operations, significantly reducing control overhead and therefore obtaining a speedup of 10.33×. The worst case occurs with the B2N2 model, the largest convolutional architecture evaluated, which experiences the highest loop control overhead. Despite this, it still achieves a significant speedup of 7.62×.

Table III presents the hardware resource utilization and power consumption of the baseline and both extended RISC-V core implementations in the last three columns. All designs proposed in this work were synthesized and implemented on the Zynq UltraScale+ ZCU104 Evaluation Board FPGA, targeting a clock frequency of 100 MHz. This frequency was chosen to match the default AXI interface clock used by the baseline processor for communication. Most existing BNN accelerators are discrete accelerators, for comparison, this work references the B2N2 [20] and ViBNN [30] accelerators in the first and second columns. The implementation data for both B2N2 and ViBNN are taken directly from the B2N2 publication, where both designs were evaluated on the same FPGA used in this work [20]. Unlike traditional discrete FPGA accelerators that rely on fixed function datapaths, this work extends a fully programmable CPU with lightweight, application-driven functional units designed for BNN inference. The third column

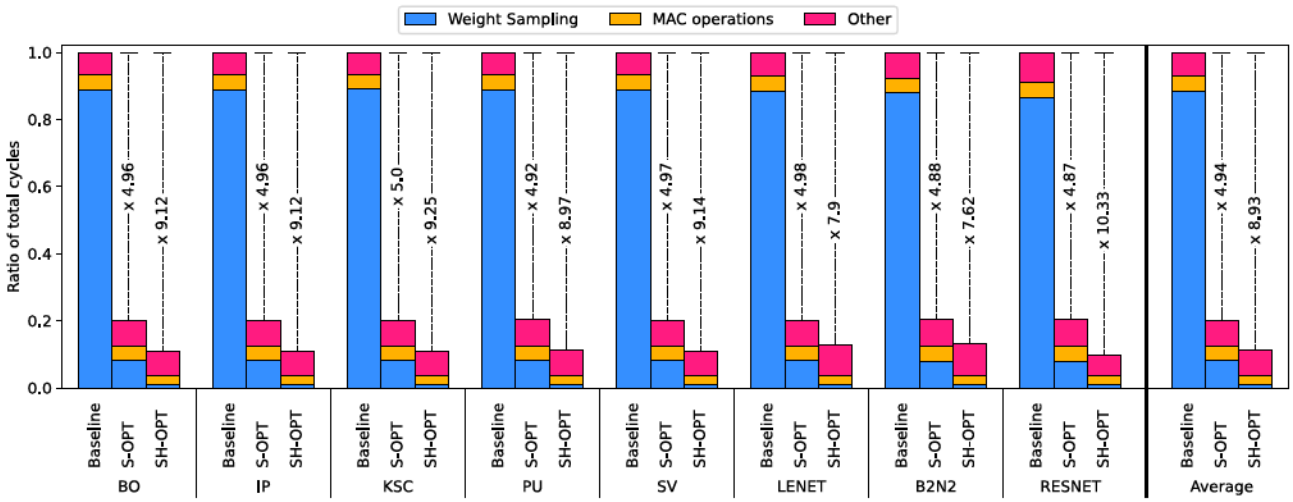

Fig. 9. Execution cycle breakdown for the test BNN models using the proposed optimizations configurations. The height of the bars represents the proportion of cycles for each category, with the speedup factors of the optimizations compared to the baseline shown above the bars.

TABLE III
FPGA IMPLEMENTATION COMPARISON OF THE PROPOSED AND PREVIOUS WORK, WITH EFFICIENCY AND THROUGHPUT EVALUATED USING THE B2N2 MODEL. THE VIBNN AND B2N2 ARE DISCRETE ACCELERATORS

Work	ViBNN [30]	B2N2 [20]	[35]	Base RISC-V	Mod. Ext.	Opt. Ext.
CLK (MHz)	300	300	100	100	100	100
# LUTs	194106	81661	2675	2224	2596	2532
# DSPs	929	465	12	12	15	12
# Registers	244436	89312	2242	1701	1745	1745
BRAM (kB)	1109	1370	54	36	36	36
Avg. FPGA U%	67.51	44.83	1.55	1.15	1.23	1.19
Power (mW)	4000	2995	48	33	38	36
Image/s	254.80	300.40	0.19	0.03	0.20	0.20
Image/J	63.70	100.30	4.05	0.78	5.18	5.47

TABLE IV
IMAGE PROCESSING EFFICIENCY (IMAGE/J) OF PROPOSED OPTIMIZATIONS

Model	Baseline	S-OPT	SH-OPT Modular	SH-OPT Optimized
BO	4998.10	24768.91	39588.69	41788.06
IP	3759.52	18665.59	29790.56	31445.59
KSC	4247.51	21240.24	34109.02	36003.96
PU	6773.37	33334.83	52767.72	55699.26
SV	3695.34	18368.26	29343.12	30973.29
LENET	1.65	8.21	11.31	11.94
B2N2	0.78	3.82	5.18	5.47
RESNET	1.95	9.47	17.46	18.43

TABLE V
FORWARD PASS THROUGHPUT (IMAGE/S) OF PROPOSED OPTIMIZATIONS

Model	Baseline	S-OPT	SH-OPT
BO	164.94	817.37	1504.37
IP	124.06	615.96	1132.04
KSC	140.17	700.93	1296.14
PU	223.52	1100.05	2005.17
SV	121.95	606.15	1115.04
LENET	0.05	0.27	0.43
B2N2	0.03	0.13	0.20
RESNET	0.06	0.31	0.66

shows the implementation cost of the previous extended RISC-V core presented for BNN acceleration [35].

The modular implementation of the extension increases LUT usage by 16.73%, register usage by 2.59%, and DSP usage by 25%. It also increases power consumption by 15.15%. The optimized implementation has less impact on the hardware cost of the design, increasing the number of LUTs only by 13.85%, not increasing the number of DSPs used, and increasing the power consumption only by 9.09%. Taking into account the speedups obtained, the modular extension results in an average energy consumption reduction of 87.12% compared to the baseline. The fully optimized version further enhances this to 87.79%. Compared to the S-OPT version, the reduction is 39.78%.

The optimized version of the extension proposed in this work increases the baseline power consumption by only 3 mW, in contrast to the B2N2 and ViBNN accelerators, which add 2995 and 4000 mW, respectively. Furthermore, the optimized functional unit requires only 308 LUTs, 44 registers and no DSP blocks, whereas the B2N2 accelerator uses 44.83% of the FPGA resources and ViBNN 63.7%. If the objective is to achieve maximum performance, the use of a dedicated accelerator paired with another CPU is the superior choice. However, when the goal is to improve performance at minimal cost, a priority for many edge systems, the proposed approach is the most cost-effective alternative, increasing efficiency while keeping costs low and preserving the versatility of a general-purpose CPU.

Taking into account these implementation cost and configuration results, Table IV presents the image processing efficiency (image/J) and Table V summarizes the throughput (image/s) of all test models comparing the proposed optimizations.

The S-OPT increases the average efficiency by a factor of 4.94×, while the optimized implementation of the extension achieves an even greater improvement, reaching an average increase of 8.19×. In a scenario where a remote battery-powered device relies on a BNN for real-time decision-making based

TABLE VI
EXECUTION CYCLES OF NNS AND BNNS IN THE TESTBENCH PROCESSOR

Model	NN	BNN Base	BNN SH-OPT
BO	50.34 K	606.29 K	66.47 K
IP	66.68 K	806.03 K	88.34 K
KSC	58.07 K	713.43 K	77.15 K
PU	37.55 K	447.39 K	49.87 K
SV	67.81 K	820.03 K	89.68 K
LENET	151.34 M	1836.84 M	232.58 M
B2N2	341.13 M	3868.86 M	508.00 M
RESNET	116.49 M	1557.29 M	150.69 M
Avg. Slowdown		12.16×	1.37×

on camera images, improvements in efficiency and throughput provide greater flexibility in system design. For example, these optimizations could enable to increase the number of forward passes, aiming to obtain a meaningful uncertainty estimate, without exceeding real-time constraints or significantly impacting battery life.

To evaluate how effectively the gap between BNNs and traditional NNs has been narrowed, the test models were executed both as classic NNs and as BNNs on the testbench processor [23], the results of these experiments are shown on Table VI. Using BnnRV without any of the proposed optimizations, on average, a BNN forward pass is 12.16× slower than an NN forward pass. With the addition of hardware and software optimizations, the performance difference is reduced to only a 1.37× average slowdown, within an small range between 1.3 and 1.5× slowdown.

VII. CONCLUSION

This paper proposes an efficient software and hardware implementation for running Bayesian Neural Networks (BNN) inference in edge devices.

On the software side, our work shows that BNN weight sampling can be optimized through the use of uniform distributions. This method delivers an average 4.94× speedup in test models without any degradation in accuracy, calibration, or the quality of uncertainty metrics.

On the hardware side, we proposed a tiny RISC-V ISA extension that leverages the software optimizations and achieves speedups between 7.62 and 10.33× over the baseline. This difference comes mainly from the ratio of the control instructions in each benchmark. More importantly, considering the small increase in power consumption with regard to the baseline processor, the extension yields a 87.79% energy consumption reduction and a 8.19× image/J efficiency increase.

In summary, this work significantly narrows the performance gap between traditional NNs and BNNs, reducing the disparity from 12.16× to only 1.37× slower. With BnnRV, on average, the weight sampling percentage goes from more than 80% to almost zero. Therefore, the weight sampling section of the algorithm stops being the performance bottleneck. Further optimization should now focus on addressing control overhead, which is now the main performance-limiting factor. For this purpose, the same optimizations used for NNs can be applied for BNNs orthogonally to the techniques presented.

REFERENCES

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[2] J. Dean, "The deep learning revolution and its implications for computer architecture and chip design," 2019, *arXiv:*1911.05289. [Online]. Available: http://arxiv.org/abs/1911.05289

[3] European Commission, "White paper on artificial intelligence: A European approach to excellence and trust," European Commission, Brussels, White Paper COM(2020) 65 final, Accessed: Feb. 20, 2020. [Online]. Available: https://commission.europa.eu/publications/white-paper-artificial-intelligence-european-approach-excellence-and-trust_en

[4] UNESCO, "Recommendation on the Ethics of Artificial Intelligence," Accessed: Nov. 23, 2021. [Online]. Available: https://www.unesco.org/en/legal-affairs/recommendation-ethics-artificial-intelligence

[5] European Innovation Council and SMEs Executive Agency, "EIC tech report 2024," European Innovation Council, Tech. Rep., Dec. 2024. [Online]. Available: https://eic.ec.europa.eu/document/6db51313-d1d5-4866-be94-ac7cee6dfb77_en

[6] S. Prakash et al., "Is Tinyml sustainable?" *Commun. ACM*, vol. 66, no. 11, pp. 68–77, Oct. 2023.

[7] A. Howard et al., "Searching for mobilenetv3," 2019, *arXiv:*1905.02244. [Online]. Available: http://arxiv.org/abs/1905.02244

[8] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019.

[9] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015, *arXiv:1506.02626*. [Online]. Available: http://arxiv.org/abs/1506.02626

[10] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," 2017, *arXiv:1712.05877*. [Online]. Available: http://arxiv.org/abs/1712.05877

[11] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," 2017, *arXiv:1704.04760*. [Online]. Available: http://arxiv.org/abs/1704.04760

[12] Apple, "iphone xs and iphone xs max bring the best and biggest displays to iphone," 2018. [Online]. Available: https://www.apple.com/newsroom/2018/09/iphone-xs-and-iphone-xs-max-bring-the-best-and-biggest-displays-to-iphone/

[13] HiSilicon, "Kirin 980 Chipset," https://www.hisilicon.com/en/products/Kirin/Kirin-flagship-chips/Kirin-980

[14] Qualcomm, "Qualcomm Artificial Intelligence Engine Powers AI Capabilities of Snapdragon Mobile Platform," 2018, [Online]. Available: https://www.qualcomm.com/news/releases/2018/02/qualcomm-artificial-intelligence-engine-powers-ai-capabilities-snapdragon

[15] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A comprehensive survey on tinyml," *IEEE Access*, vol. 11, pp. 96892–96922, 2023.

[16] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," 2017, *arXiv:1706.04599*.

[17] Z. Ghahramani, "Probabilistic machine learning and artificial intelligence," *Nature*, vol. 521, no. 7553, pp. 452–459, May 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14541.

[18] R. Krishnan and O. Tickoo, "Improving model calibration with accuracy versus uncertainty optimization," 2020, *arXiv:2012.07923*.

[19] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," 2015, *arXiv:1505.05424*.

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

[21] Ranganath Krishnan, Pi Esposito, and M. Subedar, "Bayesian-torch: Bayesian neural network layers for uncertainty estimation," 2022, doi: 10.5281/zenodo5908307.

[22] S. Pérez-Pedrajas, J. Resano, and D. Suárez-Gracia, "BNN4C," 2015. [Online]. Available: https://github.com/samuprpd/BNN4C

[23] S. Pérez-Pedrajas, J. Resano, and D. Suárez-Gracia, "GaZmusino," 2019. [Online]. Available: https://github.com/samuprpd/GaZmusino

[24] YosysHQ, "picorv32," 2019. [Online]. Available: https://github.com/YosysHQ/picorv32

[25] S. Ahmadi-Pour, V. Herdt, and R. Drechsler, "The microrv32 framework: An accessible and configurable open source RISC-V cross-level platform

for education and research," *J. Syst. Archit.*, vol. 133, 2022, Art. no. 102757.

[26] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "Mr. Wolf: An energy-precision scalable parallel ultra low power soc for IoT edge processing," *IEEE J. Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, Jul. 2019.

[27] M. Gautschi et al., "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Trans. Very Large Scale Integr.*, vol. 25, no. 10, pp. 2700–2713, Oct. 2017.

[28] A. Garofalo, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, "XpulpNN: Enabling energy efficient and flexible inference of quantized neural network on RISC-V based IoT end nodes," Nov. 2020.

[29] V. Potocnik et al., "Optimizing foundation model inference on a many-tiny-core open-source RISC-V platform," *IEEE Trans. Circuits Syst. Artif. Intell.*, vol. 1, no. 1, pp. 37–52, Jan. 2024.

[30] R. Cai et al., "VIBNN: Hardware acceleration of Bayesian neural networks," 2018, *arXiv:1802.00822*.

[31] Y. Hirayama, T. Asai, M. Motomura, and S. Takamaeda, "A hardware-efficient weight sampling circuit for Bayesian neural networks," *Int. J. Netw. Comput.*, vol. 10, no. 2, pp. 84–93, 2020. [Online]. Available: http://www.ijnc.org/index.php/ijnc/article/view/222

[32] H. Awano and M. Hashimoto, "Bynqnet: Bayesian neural network with quadratic activations for sampling-free uncertainty estimation on FPGA," in *Proc. Des. Automat. Test Eur. Conf. Exhib. (DATE)* 2020, pp. 1402–1407.

[33] X. Jia, J. Yang, R. Liu, X. Wang, S. D. Cotofana, and W. Zhao, "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, Apr. 2021.

[34] X. Jia et al., "An energy-efficient Bayesian neural network implementation using stochastic computing method," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 9, pp. 12913–12923, Sep. 2024.

[35] S. Pérez, J. Resano, and D. S. Gracia, "Accelerating Bayesian neural networks on low-power edge RISC-V processors," in *Proc. IEEE 24th Int. Conf. Nanotechnol. (NANO)*, 2024, pp. 507–512.

[36] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/

[37] A. Ansel et al., "Pytorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation," in *Proc. 29th ACM Int. Conf. Architect. Supp. Program. Lang. Oper. Syst.*, vol. 2. New York: ACM, Apr. 2024. [Online]. Available: https://Pytorch.org/Assets/Pytorch2-2.pdf

[38] J. V. Dillon et al., "Tensorflow distributions," 2017, *arXiv:1711.10604*. [Online]. Available: http://arxiv.org/abs/1711.10604

[39] J.-L. Lin et al., "Quantization for Bayesian deep learning: Low-precision characterization and robustness," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*. Piscataway, NJ, USA: IEEE Press, 2023, pp. 180–192.

[40] T. Chen et al., "TVM: End-to-end optimization stack for deep learning," 2018, *arXiv:1802.04799*. [Online]. Available: http://arxiv.org/abs/1802.04799

[41] M. Pakdaman Naeini, G. Cooper, and M. Hauskrecht, "Obtaining well calibrated probabilities using Bayesian binning," *Proc. AAAI Conf. Artif. Intell.* vol. 29, no. 1, 2015. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/9602

[42] A. Alcolea and J. Resano, "Bayesian neural networks to analyze hyperspectral datasets using uncertainty metrics," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, pp. 1–10, 2022, Art. no. 5537810.

[43] C. E. Shannon, "A mathematical theory of communication," *Bell System Tech. J.*, vol. 27, pp. 379–423, 1948. [Online]. Available: http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

[44] J. Mukhoti and Y. Gal, "Evaluating Bayesian deep learning methods for semantic segmentation," 2018, *arXiv:1811.12709*.

[45] M. Laves, S. Ihler, K. Kortmann, and T. Ortmaier, "Well-calibrated model uncertainty with temperature scaling for dropout variational inference," 2019, *arXiv:1909.13550*, 2019. [Online]. Available: http://arxiv.org/abs/1909.13550

[46] R. Krishnan, M. Subedar, and O. Tickoo, *"Specifying weight priors in Bayesian deep neural networks with empirical Bayes*," 2020, *arXiv:1906.05323*. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/5875

[47] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665.

[48] A. Krizhevsky, "Learning multiple layers of features from tiny images," Comput. Sci. Dept., Univ. Toronto, Tech Rep., 2009, pp. 32–33. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[49] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 123–147.

[50] C. Banbury et al., "MLPerf tiny benchmark," 2021, arXiv:2106.07597.

[51] Embench, "embench-IoT," 2020. [Online]. Available: https://github.com/embench/embench-iot

[52] G. Marsaglia, "Xorshift RNGs," *J. Statist. Softw.*, vol. 8, no. 14, pp. 1–6, 2003. [Online]. Available: https://www.jstatsoft.org/index.php/jss/article/view/v008i14

[53] L. Colavito and D. Silage, "Efficient PGA LFSR implementation whitens pseudorandom numbers," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, 2009, pp. 308–313.