BNRV: A Lightweight SIMD Extension for Efficient BitNet Inference on RISC-V CPUs

Zijun Jiang and Yangdi Lyu†
Microelectronics Thrust, The Hong Kong University of Science and Technology (Guangzhou)
†Corresponding author: yangdilyu@hkust-gz.edu.cn

***Abstract*—AI models utilizing extremely low-bitwidth weights have shown promise in efficient storage and computation while maintaining satisfactory results through proper training processes. By converting floating-point multiplication into simpler addition and shifting operations, these models are well-suited for deployment on resource-constrained devices. However, traditional CPU architectures often fail to fully exploit the advantages of multiplication-free operations due to a lack of hardware support.**

In this paper, we propose BNRV, a lightweight SIMD extension designed for the RISC-V Instruction Set Architecture (ISA) that specifically targets multiplication-free operations involving 8-bit data and low-bitwidth weights (ranging from 1 to 2 bits). We have also developed an accompanying library with optimized kernels for deploying various AI models using BNRV. Our proposed extension significantly accelerates low-bitwidth quantized multiplication (up to 10.95× times faster) and low-bitwidth transformer model inference (up to 3.11× faster), with minimal power overhead (less than 2%) and area overhead (less than 4%) compared to processors without BNRV support.

***Index Terms*—RISC-V, SIMD, TinyML, Quantization**

I. Introduction

Quantized Neural Networks (QNN) have emerged as a key technology for enabling neural network inference on devices with strict power and memory constraints. By reducing the precision of weights and activations, QNNs significantly decrease the storage and computational requirements, making them suitable for deployment on mobile and embedded platforms. Low-bitwidth quantization, such as binary and ternary weight representations, further improves the efficiency by simplifying the arithmetic operations in neural network computations, especially matrix multiplications that are the cornerstone of deep learning workloads.

Extensive research has demonstrated the superior efficiency of low-bitwidth quantization in convolutional neural networks (CNNs) [1]–[3] and transformer-based large language models (LLMs) [4], [5]. These studies show that even binary or ternary quantized LLMs can deliver decent results with a proper training process, while significantly reducing memory and computational requirements. Besides, the absence of multiplication in low-bitwidth quantized inference presents an opportunity to deploy compact language models on resource-constrained edge devices at a low cost and with reduced power consumption.

However, the inference of low-bitwidth quantized networks on conventional CPUs suffers from inefficiencies due to inadequate hardware support. While existing accelerators [6]–[8] can effectively accelerate the inference, they primarily focus on binarized networks and do not support emerging formats like ternary weights. Moreover, these accelerators often have significant area overhead as standalone units, making them impractical to integrate into low-end microcontrollers (MCUs). A promising alternative is to extend general-purpose CPUs with custom instruction sets, which can provide acceleration with lower overhead. Single Instruction Multiple Data (SIMD) instruction extensions are commonly used to accelerate data processing on CPUs. However, traditional SIMD architectures are not optimized for extremely low-bitwidth operations. For example, the RV32P Packed SIMD extension only supports 8-bit and 16-bit arithmetic operations.

To address the challenges posed by the deployment of low-bitwidth quantized models on conventional CPUs, we propose **BNRV**, a lightweight ISA extension and hardware design for RISC-V CPUs. The BNRV extension enables significant performance gains for extremely low-bitwidth quantized models without imposing substantial power or area penalties.

The main contributions of this paper are as follows:

- We propose a lightweight SIMD extension for the RISC-V ISA that specifically targets multiplication-free operations between 8-bit data and low-bitwidth (1 to 2 bits) weights. This extension, available in two versions (buffer-free and buffered), requires only 1 or 2 instructions, respectively, enabling parallel processing of 4 or 8 operations. The extension speeds up the matrix multiplication operation of low-bitwidth quantized neural networks by up to 10.95×.
- We design a low-overhead Register-transfer Level (RTL) implementation of the proposed extension, which incurs less than 2% power overhead and less than 4% area overhead compared to a standard CPU. The implementation is highly configurable, enabling the generation of various BNRV designs with different weight widths and buffer sizes.
- We develop a library with optimized BitNet kernels utilizing BNRV[1], and deploy multilayer perceptron (MLP)-based models, CNN models and transformer-based language models with the library. With the extension and the library, the BitNet-based transformer model inference can be accelerated by up to 3.11× on extended RV32IMC CPUs, and by up to 4.37× on extended RV32IMFC CPUs.

[1]github.com/HKUSTGZ-MICS-LYU/BNRV

II. Background

A. Low-bitwidth Quantization

Fully connected or linear layers are heavily adopted in deep neural networks (DNNs). In regular linear layers, the computation between the input $x \in \mathbb{R}^{1\times d}$ and the weight matrix $W \in \mathbb{R}^{m\times d}$ can be expressed in the form of matrix multiplication (MatMul):

$$y = xW \tag{1}$$

where $y \in \mathbb{R}^{1\times m}$ is the output, and i-th element in vector y is computed as:

$$y_i = \sum_{j=1}^{d} x_j W_{ij} \quad (i = 1, \ldots, m) \tag{2}$$

The MatMul computations are typically carried out using floating-point (FP) data, which leads to an enormous number of floating-point multiplications. The *quantization* technique addresses the high computational requirement by converting the FP data into integers with reduced bitwidth (e.g., 8-bit integer (INT8)). This approach decreases the computational, storage, and memory needs of models, facilitating the deployment of models on edge devices.

To further reduce the computational burden of floating-point or integer multiplications, BitNet [4] introduces BitLinear layers with binary (1-bit), ternary (1.58-bit), or quaternary (2-bit) weights. In this framework, binary weights are limited to values of $\{+1, -1\}$, ternary weights introduce an additional value 0, and quaternary weights introduce -2. The weights of neural networks are quantized from 32-bit floating-point representations down to these 1-bit or 2-bit formats to represent the small sets of values, while the activation is quantized to 8-bit integers. By using the quantized weights $\widetilde{W}_{ij}$, the multiplications in MatMul are replaced with simple additions and subtractions. For example, the multiplications with ternary quantized weights can be represented as:

$$x_j \widetilde{W}_{ij} = \begin{cases} x_j, & \text{if } \widetilde{W}_{ij} = 1, \\ 0, & \text{if } \widetilde{W}_{ij} = 0, \\ -x_j, & \text{if } \widetilde{W}_{ij} = -1, \end{cases} \tag{3}$$

Therefore, the BitNet MatMul can be computed using simple hardware without any multiplier. The 2-bit quantization, which introduces an additional term $-2x_j$, can be implemented with a bit shifting operation, i.e., $-2x_j = (-x_j) \ll 1$.

B. BitLinear Layer in BitNet

The basic architecture of a BitLinear layer is illustrated in Fig. 1. Both the inputs and outputs of a BitLinear layer are in floating-point format (FP32). Before the BitNet MatMul, the FP32 inputs are first normalized using root mean square normalization (RMSNorm) and are subsequently quantized to 8-bit integers based on their absolute maximum values. After the quantized inputs are "multiplied" with the 1- or 2-bit weights using the BitNet MatMul operation, the results need to be de-quantized back to FP32 outputs to restore precision. This dequantization process is done by multiplying the scaling factors of activations and weights (s_a and s_w).

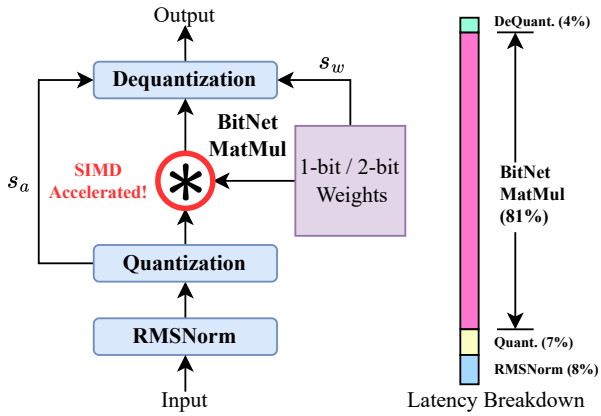

Fig. 1: The Basic Architecture and Latency Breakdown of a BitLinear on a Conventional CPU

Fig. 1 also shows the latency breakdown of a BitLinear layer profiled on an RV32IMC CPU, revealing that the BitNet MatMul operation accounts for more than 80% of the total execution time. Despite the easy-to-compute nature of the BitNet MatMul, the operation becomes the bottleneck on regular CPUs due to the lack of native support for low-bitwidth operations, which motivates us to extend the ISA and unleash the full potential of BitNet-based models.

C. BitNet-based Language Model

Conventional language models are mostly based on Transformers [9], which typically contain many linear layers with associated weight matrices. This architecture results in high computational demands for standard MatMuls. This issue can be effectively mitigated by substituting these layers with BitLinear layers.

In order to investigate the impact of quantization widths on loss, we trained small-scale BitNet-based transformer models on the TinyStories dataset [10], with the experimental results shown in Fig. 2. The results demonstrate that while the 2-bit quantized BitNet models achieve the lowest loss values, the gap between the 1.58-bit and 2-bit models keeps shrinking as the model size increases. Subsequent sections will elaborate on how varying quantization widths influence hardware complexity and model storage sizes, which necessitates a comprehensive consideration of width choices.

D. RISC-V SIMD Extension

To accelerate the QNN inference on resource-constrained CPUs, many existing works have proposed to extend the RISC-V ISA with SIMD extensions for common computations in neural networks.

The official RISC-V Packed SIMD Extension [11] provides general-purpose SIMD operations primarily for 8-bit and 16-bit data types, including addition, subtraction, and multiplication. However, its architecture is not optimized for the ultra-low-bitwidth weights characteristic of BitNet models. Using

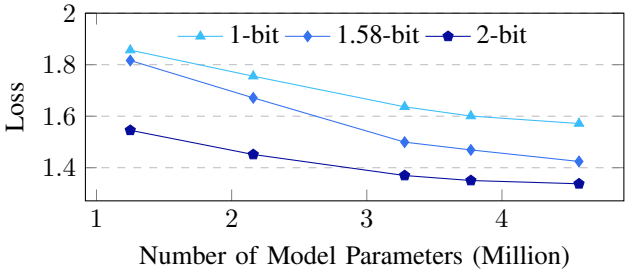

Fig. 2: Scaling Law of Small-scale BitNet Models

this extension to accelerate BitNet would necessitate extensive software-based masking and unpacking to convert weights to a compatible 8-bit format. This pre-processing overhead, as demonstrated by comparisons with generic 8-bit dot product approaches, significantly degrades performance, rendering it unsuitable for efficient BitNet inference.

Other notable extensions, such as XPulpNN [12] for 2/4-bit operations and XTern [13] for ternary weights, target scenarios with uniform quantization where both activations and weights share the same low precision. This paradigm does not align with BitNet, where 8-bit activations are combined with weights ranging from 1 bit to 2 bits, creating a gap in hardware acceleration for such mixed-precision operations.

MPIC [14] is another extension aimed at mixed precision, supporting operations between operands with different bitwidths. Nonetheless, it only supports weights down to 2-bit, while binary and ternary weights are neglected. Besides, its implementation incurs a larger area overhead of 11% on the CPU core.

III. The BNRV SIMD Extension

To address the need for SIMD computation with 8-bit activations and low-bitwidth weights, we propose an extension in lightweight multiplier-free hardware that can be easily integrated into resource-constrained edge devices.

A. Weight Representation

For binary and quaternary weights, 1-bit and 2-bit formats, respectively, are optimal for storage. While it is feasible to compress 5 ternary weights into an 8-bit string using a specific encoding scheme [15], additional hardware and software decompression mechanisms are needed. To minimize the hardware overhead, we store all ternary weights (1.58-bit) in a 2-bit format. Despite using a 2-bit storage format, we refer to ternary weights as 1.58-bit throughout the paper to clearly differentiate them from quaternary weights.

B. Basic Operation Units

TABLE I: Output Selections for Different Quantization

Quaternary 2-bit				
Ternary 1.58-bit				
Binary 1-bit				
Weight	1	-1	0	-2
Output	X	$-X$	0	$-2X$

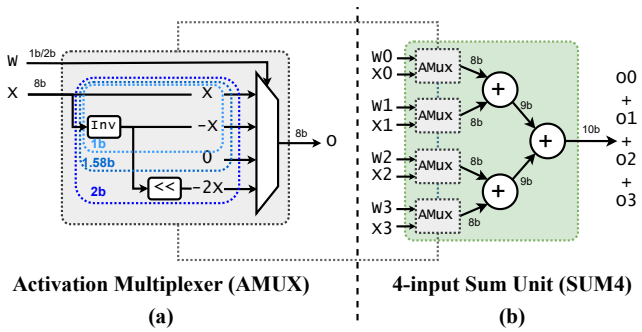

Fig. 3: Hardware Architecture of BNRV Units

To convert the multiplication into simple operations for extremely low-bitwidth (binary, ternary, or quaternary) BitNet-based models, we design the core function unit of our BNRV SIMD extension, as depicted in Fig. 3.

1) Activation Multiplexer (AMUX): Fig. 3(a) illustrates the architecture of an activation multiplexer, which performs the BitNet "multiplication" between an 8-bit input X and a 1-bit or 2-bit weight W. The data paths for different quantization formats are highlighted in different blue squares. For 1-bit weights representing binary values $\{1, -1\}$, a signed integer inverter is added alongside the direct path from the 8-bit input. To support 1.58-bit weights with an additional value of 0, we incorporate a constant zero into the multiplexer. For 2-bit weights that introduce an additional value of -2, we add a path with a 1-bit shift left operation to generate $-2X$. The final output value of AMUX is selected based on the value of the 1-bit or 2-bit weight, as indicated in Tab. I. Given that these quantization types have overlapping value ranges, the data paths within the AMUX can be effectively reused, thereby minimizing overall hardware overhead.

As we can see, the AMUX, serving as the core computing component for the "multiplication" operation in BitNet, does not require resource-intensive operators like multipliers, which lays the foundation for the lightweight feature of the BNRV extension.

2) 4-input Sum Unit (SUM4): To enable the SIMD processing of BitNet MatMul operations, we design a 4-input sum unit that performs dot product computations with an activation vector and a weight vector, as shown in Fig. 3(b). Each SIMD operation processes 4 inputs, resulting in a 32-bit activation vector (4×8-bit) and a weight vector that can be either 4 bits or 8 bits (4×1-bit/2-bit). The unit first passes the elements from the vectors to four AMUXs, generating four 8-bit outputs. These outputs are then summed using an adder tree to produce the final output. The results of the dot product serve as partial sums for the BitNet MatMul operations.

C. SIMD Instructions and Implementations

The BNRV ISA extension is a 32-bit RISC-V ISA extension, in which the registers are assumed to be 32-bit. It utilizes a packed SIMD approach to pack multiple activation data (8-

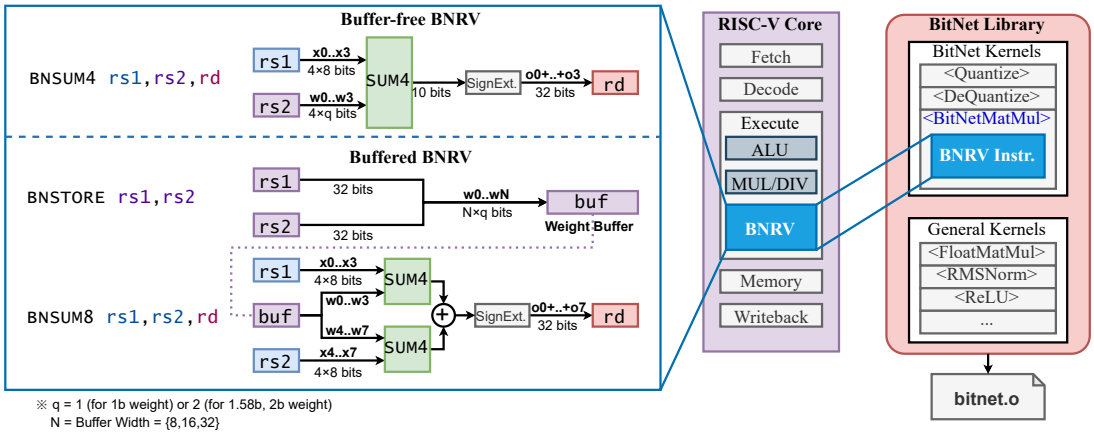

Fig. 4: Overview of BNRV Extension and Implementation.

bit) or weight data (1-bit/2-bit) into 32-bit registers for parallel computation using the basic units mentioned in the previous section. The extension is compact yet powerful, supporting all 3 types of low-bitwidth weight quantization. Fig. 4 provides an overview of the BNRV extension and its implementation.

As shown in Fig. 4, there are two implementation variants: buffer-free and buffered, each offering a trade-off between hardware overhead and performance. The different variants of BNRV extensions are denoted as BNRV4 for the buffer-free version and BNRV8/16/32 for the buffered version, which differ in the number of weights held in the buffer.

1) Buffer-free Version: The buffer-free version of BNRV is designed to minimize overhead while improving performance by utilizing a single instruction: `BNSUM4`, which is illustrated in the upper part of Fig. 4.

The instruction is an R-type instruction that operates on two register values (`rs1` and `rs2`) from the register file of the CPU. Register `rs1` holds four 8-bit activation values, and register `rs2` contains four 1-bit or 2-bit quantized weights in its lower bits. These values are passed to a SUM4 unit (Fig. 3(b)) to compute the 4-element dot product result, which is then sign extended to 32 bits and written back to register `rd`.

2) Buffered Version: A key limitation of the buffer-free version is that only the lower 4 or 8 bits of the 32-bit register `rs2` are used. To address this limitation, the buffered version separates the computation into two instructions: `BNSTORE` and `BNSUM8`, which are illustrated in the lower part of Fig. 4.

The `BNSTORE` instruction is responsible for storing weight data in a dedicated weight buffer `buf`. It reads weight data from both registers `rs1` and `rs2`, and stores them into the weight buffer. This weight buffer can hold 8, 16, or 32 weights, corresponding to the BNRV8, BNRV16, and BNRV32 configurations, respectively.

The second instruction, `BNSUM8`, performs an 8-element dot product using the activation values from `rs1` and `rs2`, along with the weight values stored in the weight buffer. For example, the weight buffer in BNRV16 accommodates 16 weights by the `BNSTORE` instruction, which can be used for two subsequent `BNSUM8` instructions. While the buffered version enhances register utilization and parallelism, it introduces additional hardware components, such as an extra SUM4 unit and a dedicated weight buffer, resulting in increased hardware overhead.

Beyond its application in BitNet MatMul operations, the BNRV extension can also be used for general 8-bit summing or subtracting by setting weights to either 1s or -1s.

D. BitNet MatMul Using BNRV Instruction

In this section, we will demonstrate the implementation of BitNet MatMul using BNRV instructions. For ease of illustration, we will use 1.58-bit quantized weights as an example. The process is similar for the other two cases.

As discussed in Sec. II-A, BitNet MatMul operations with 1.58-bit quantized weights can be expressed as:

$$\widetilde{y}_i = x \circledast \widetilde{W} = \sum_{j:\widetilde{W}_{ij}=1} x_j - \sum_{j:\widetilde{W}_{ij}=-1} x_j \tag{4}$$

where $\circledast$ denotes the BitNet MatMul that can be computed using simple operations without multiplication. To implement this kernel function utilizing BNRV instructions, we integrate assembly functions into the hot loop of BitNet MatMul. Fig. 5 shows an example usage of how BNRV instructions are used in the BitNet MatMul kernel function with 1.58-bit weights.

For CPUs without BNRV, additional masking operations are required to unpack the 1.58-bit weights (`w`) from the 8-bit weight packages (`w4`), and two conditional operators are used to get the result. For CPUs with buffer-free BNRV (BNRV4), four 8-bit inputs and one weight package are directly passed to the custom instruction `BNSUM4`, and these four pairs of inputs and weights are processed in parallel. With buffered BNRV (BNRV8), an additional buffer storing instruction `BNSTORE` is used to store 16-bit data (eight 1.58-bit weights) in the weight buffer, allowing for the simultaneous processing of 8 pairs of data using `BNSUM8`.

We further investigate the instruction counts of the hot loop by compiling the code into assembly. Without BNRV, it takes around 18 instructions for a single computation in the hot

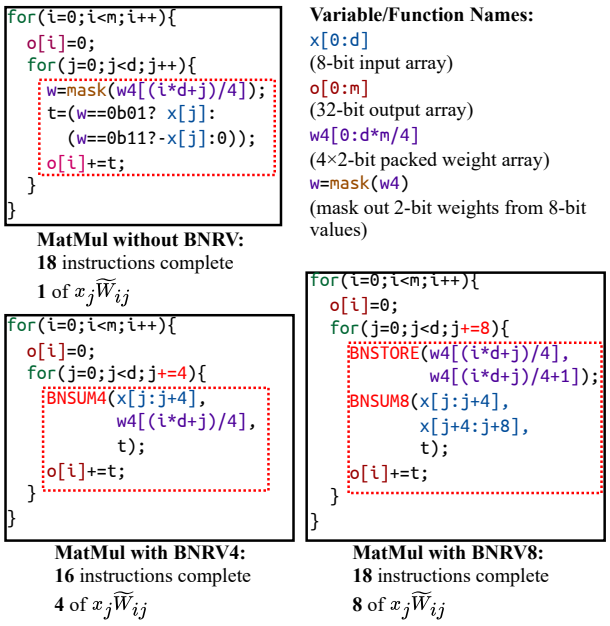

Fig. 5: Examples of BitNet MatMul with BNRV4 and BNRV8

loop, resulting in a total of $18 \times m \times d$ instructions for a complete MatMul. In contrast, with BNRV4, the instruction count is significantly reduced to 16 instructions for four computations, which is 4 instructions per computation. The approximate reduction ratio for BitNet MatMul with BNRV4 is thereby $18/4 = 4.5$. The instructions per computation are further reduced using BNRV8, with 18 instructions for eight computations. It achieves a reduction ratio of $18/2.25 = 8$.

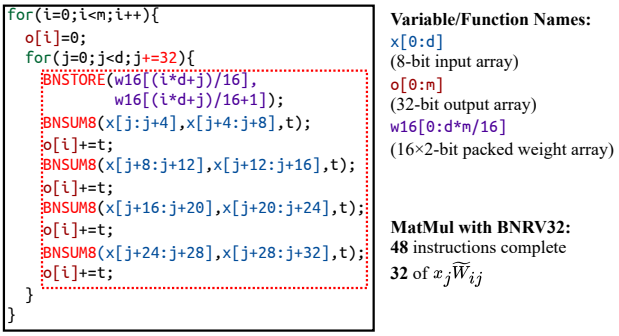

Fig. 6: Example of BitNet MatMul with BNRV32

With BNRV32, which extends the weight buffer to store 32 elements, the BitNet MatMul implementation can be further improved, as shown in Fig. 6. Following a `BNSTORE` that loads 32 weights into the buffer, four `BNSUM8` instructions are executed sequentially to compute the partial results. With BNRV32, we can complete 32 computations using just 48 instructions, achieving a reduction ratio of $18/1.5 = 12$, resulting in the highest speedup.

E. BitNet Library Support

As shown in Fig. 4, we implement a library in C and assembly code to support end-to-end inference of BitNet-based models. For the core kernel BitNet MatMul, we provide both a generic version and a SIMD-accelerated version. The generic version, which runs on basic CPUs, unpacks the quantized weights to 8-bit and computes the MatMul element by element. For the SIMD-accelerated version, we build assembly intrinsics as mentioned in Sec. III-D, and optimize BitNet MatMul by calling the assembly intrinsics (similar to Fig. 5).

In addition to the BitNet MatMul, we also implement other essential kernels, such as quantization and de-quantization, to support a full BitLinear layer. While these floating-point kernels are not accelerated by the BNRV extension, they typically account for a very small portion of inference time (as shown in Fig. 1). Furthermore, we implement convolution layers with the BitNet MatMul function to support quantized CNNs via the widely used Im2Col+MatMul computation, which can be accelerated by BNRV as well.

F. Quantization, Training and Deployment

To effectively utilize the BNRV extension, a framework is required to train quantized models and deploy the BitNet-based models on the proposed hardware. However, many existing deployment frameworks, such as TVM [16] and DORY [17], primarily focus on 8-bit quantized models, while the emerging quantization methods like BitNet are not supported.

Therefore, we implement our own deployment framework for efficient BitNet deployment on the proposed BNRV hardware. The details of the framework are described as follows.

1) Quantization: The models are layer-wise quantized, with a floating-point scaling factor s_w saved for each BitLinear layer. For 1-bit quantization, weights are first centralized to have zero mean and then binarized with the signum function [4]. For 1.58-bit and 2-bit quantization, weights are quantized using the absolute mean quantization function [5].

2) Training: We use the quantization-aware training (QAT) technique for model training. The straight-through estimator (STE) is used to bypass non-differentiable quantization functions. Training states, including gradients, are stored in high precision (32-bit floating point) to ensure training stability, according to [4].

3) Deployment: To deploy the BitNet-based model onto RISC-V CPUs with or without our BNRV extension, we develop a dedicated code generator that directly generates bare-metal C code from PyTorch-based BitNet models, integrating the aforementioned BitNet library. The model weights in 1-to-2-bit format will be exported and linked with the bare-metal executable.

IV. Experimental Result

A. Experiment Setup

We extended the open-source RISC-V CPU VexiiRiscv [18] by incorporating the proposed BNRV extension as one of its

execution stage components. We used Verilator-based cycle-accurate RTL simulation to evaluate the inference latency of BitNet-based models.

B. Hardware Impact

We synthesized the RISC-V CPU with the proposed extension with Synopsys Design Compiler-2023.12 using the ASAP 7nm technology [19] to analyze the area and power consumption. The baseline design selected is a minimal Vexi-iRiscv CPU supporting RV32IMC ISA, which is a 5-stage in-order CPU design without any extra components like caches or branch predictors. The core frequency is set to 500 MHz. Notably, the simplicity of the BNRV extension, which employs a compact and multiplier-free design, ensures that it does not introduce any new critical paths in the design, allowing the core frequency to remain unchanged despite the addition of the extension.

We first evaluate the hardware impact of different BNRV extensions. The hardware overhead of different variants is listed in Tab. II. Compared to the baseline CPU, the buffer-free BNRV extension (BNRV4) only introduces a 1.29% increase in area and negligible power overhead. For the buffered BNRV extensions, the overhead rises to 3.85% in area and 1.53% in power as the buffer width increases, which is still very small considering its acceleration capability, which will be discussed in the following sections.

TABLE II: Hardware Impact of Different BNRV

Extension	Area (μm^2)	Power (mW)	Perf. (GOPs)
RV32IMC	38782	1.90	0.04 (1.00×)
+BNRV4	39281 (+1.29%)	1.90 (+0.00%)	0.14 (3.87×)
+BNRV8	39666 (+2.28%)	1.90 (+0.00%)	0.26 (7.20×)
+BNRV16	40004 (+3.15%)	1.91 (+0.51%)	0.29 (7.95×)
+BNRV32	40274 (+3.85%)	1.93 (+1.53%)	0.39 (10.95×)

To evaluate the speedup of BNRV variants, we benchmarked the performance on the hardware with a $128\times128\times128$ BitNet MatMul kernel, which is listed in the last column of Tab. II. As shown in the Tab. II, the performance of the original base CPU for BitNet MatMul operation is very undesirable due to the unpacking overhead. However, with BNRV extensions designed for this type of operation, the performance is largely boosted with subtle overhead, **accelerating BitNet MatMul operations by 3.87× to 10.95×**. Although the actual speedup is slightly lower than the theoretical speedup in Sec. III-D due to the architectural factors like pipeline stalls, the overall scaling aligns well with our expectations.

Next, we evaluate the hardware impact of the various supported quantization types. The area and power overhead compared to the RV32IMC baseline CPU are listed in Tab. III. As we expand support from only 1-bit weights to all three types of weights, the overhead of BNRV grows slightly. Specifically, the overhead of 1.58-bit and 2-bit weights is greater than that of 1-bit weights in the buffered BNRV32, as they need a 64-bit weight buffer to hold 32 weights, while the 1-bit only needs one 32-bit register.

TABLE III: Hardware Impact of Different Quantization

BNRV	Quant. Type	+Area (μm^2)	+Power (mW)
BNRV4	1-bit	360 (0.93%)	<0.01 (0.00%)
	1.58-bit	485 (1.25%)	<0.01 (0.00%)
	2-bit	499 (1.29%)	<0.01 (0.00%)
BNRV32	1-bit	1050 (2.71%)	0.01 (0.51%)
	1.58-bit	1501 (3.87%)	0.03 (1.66%)
	2-bit	1492 (3.85%)	0.03 (1.53%)

C. Evaluation on BitNet-based Models on MNIST

To evaluate the speedup provided by the BNRV extension, we first implemented and deployed a BitNet-based MLP model and a CNN model trained on the MNIST dataset. The MLP model consists of four cascaded linear layers and processes a 16×16 input image which is flattened into a 1×256 vector. The CNN model consists of two convolution layers and three linear layers, which follow the classic LeNet5 architecture [20]. Both full precision and quantized models were trained for 10 epochs using the same learning rate of 0.001. We compared the BitNet-based models with the full precision models and 8-bit integer quantized models, with the results shown in Tab. IV.

TABLE IV: Comparison of Quantized Models

Model	Precision	Accuracy (%)	Size (KB)	Comp. Ratio
MLP	FP32	97.9	100.3	1.00×
	INT8	97.9	26.4	3.80×
	INT1.58	95.9	**8.0**	**12.54**×
LeNet	FP32	99.1	120.8	1.00×
	INT8	99.1	32.6	3.71×
	INT1.58	97.7	**9.3**	**12.99**×

As shown in the table, with BitNet-based ternary quantization, the models are compressed at a ratio of over 12× compared to the original full precision models, demonstrating the superior memory efficiency of the quantized models. With QAT to maintain the accuracy, the ternary quantized MLP model and LeNet model exhibited only a 1.4% to 2.0% accuracy drop on the MNIST dataset, despite the high compression ratio.

Using the deployment flow described in Sec. III-F, we deployed the model on both the standard RV32IM[F]C CPU and the BNRV extended CPU to evaluate the latency of inferences. The latency results are shown in the Tab. V.

TABLE V: Inference Latency of Quantized Models

Model	Precision	Hardware	MatMul		Overall	
			Time	Speedup	Time	Speedup
MLP	FP32	RV32IMC	6.14 ms	1.00×	6.16 ms	1.0×
	INT8	RV32IMC	0.56 ms	10.96×	0.86 ms	7.16×
	INT1.58	RV32IMC	1.42 ms	4.32×	1.73 ms	3.56×
	INT1.58	+BNRV4	**0.36 ms**	**17.16**×	**0.67 ms**	**9.19**×
	INT1.58	+BNRV32	**0.13 ms**	**47.23**×	**0.44 ms**	**14.00**×
LeNet	FP32	RV32IMFC	16.49 ms	1.00×	17.20 ms	1.00×
	INT8	RV32IMFC	9.77 ms	1.69×	17.99 ms	0.96×
	INT1.58	RV32IMFC	25.53 ms	0.66×	33.75 ms	0.48×
	INT1.58	+BNRV4	**6.22 ms**	**2.65**×	**14.38 ms**	**1.20**×
	INT1.58	+BNRV32	**2.28 ms**	**7.23**×	**10.49 ms**	**1.64**×

As shown in the upper part of Tab. V, we compare the latency of the MLP model with different precision on the basic RV32IMC CPU and its BNRV extended version. Due to a lack of floating-point computation capability, the base

CPU is extremely slow in full precision MatMuls, and INT8 quantization resolves the intense FP operations, leading to a 7.16× overall speedup. However, when ternary quantization is applied, while the model size is further reduced with ternary quantization, the overall speedup decreases to 3.56× due to the requirement of unpacking ternary weights at runtime. Integrating with our proposed BNRV extension, the MatMul operation can be further accelerated by 17.16× to 47.23× for the 1.58-bit models compared to the FP32 models, resulting in a better overall speedup of 9.19× to 14×.

For the more computation-intensive LeNet model, we used the RV32IMFC CPU with a floating-point unit (FPU) as the base design, and extended it with BNRV extension as well. The latency comparison is shown in the lower part of Tab. V. On the base CPU with floating-point hardware, latencies for the FP32 model and INT8 model are similar, but the ternary model remains slower. Extending the more powerful CPU with BNRV, we achieve an overall speedup of 1.20× to 1.64× for the ternary model compared to the FP32 model.

Compared to the ternary model inference on basic designs (INT1.58 on RV32IMC), which requires weight unpacking, **the BNRV-extended designs achieve up to** 11.19× **faster BitNet MatMul performance, and** 3.93× **faster BitNet-based model inference**, unleashing the full potential of the emerging quantization techniques.

D. Evaluation on BitNet-based Language Model

TABLE VI: Architecture of Tiny LLaMa2 for TinyStories

Parameter	Value	Parameter	Value
Vocabulary Size	512	Sequence Length	512
Embedding Dimension	256	Hidden Dimension	256
# of Layers	8	# of Attention Heads	8
# of KV Heads	4	Dropout Ratio	0.05
Total # of Parameters:			**3,276,800**

We implemented a BitNet-based language model by substituting the linear layers in the transformer blocks of LLaMa2 [21] with BitLinear. We chose not to replace or quantize the output layer, as this led to significant performance degradation in our evaluations. Consequently, the output layer weights are maintained as FP32 values, and the standard FP32 MatMul (FMatMul) is used to compute the final outputs. The architecture parameters are listed in Tab. VI.

After training the model for 400,000 epochs (approximately 24 hours) on a single RTX 3080 GPU with 10 GB VRAM, the model results are listed in Tab. VII. When utilizing FP32 or INT8 weights, the model sizes are 12.5 MB and 3.4 MB, respectively, which are typically too large for resource-constrained edge devices like the STM32H742 microcontroller, which has only 2 MB of flash storage. In contrast, the BitNet-based model is much smaller at just 1.3 MB. Although this reduction in model size results in an accuracy drop of 8.18%, the trade-off is considered acceptable given the impressive compression ratio, particularly in resource-constrained environments.

We deployed the trained model by integrating the open-source C implementation of LLaMa2 [22] and our BitNet library. We evaluated the average latency per token on VexiiRiscv CPUs with RV32IMC, and the speedups are shown in Fig. 7. On the CPU without BRNV extension, the BitNet MatMul accounts for over 70% of the overall running time. However, with the proposed extension, the operation is significantly accelerated, resulting in an end-to-end inference speedup of up to 3.11×.

TABLE VII: Comparison of Tiny LLaMa2 Models

Precision	Loss	Accuracy (%)	Size (MB)	Comp. Ratio
FP32	0.781	77.01	12.5	1.00×
INT8	0.794	76.79	3.4	3.67×
INT1.58	1.102	68.83	**1.3**	**9.62**×

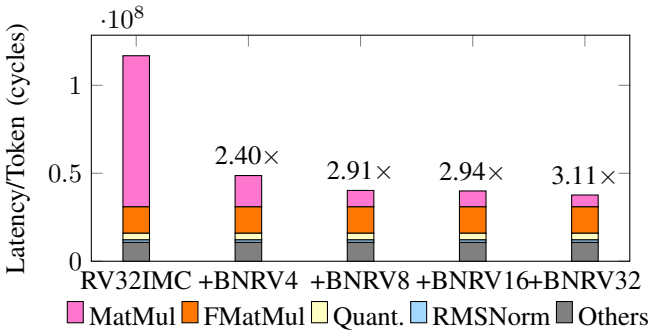

Fig. 7: BitNet-based LLaMa-2 Speedup

With the BitNet MatMul greatly accelerated by BNRV, FP operations (e.g., FMatMul) become the bottleneck. To address this, we conducted the same experiment on a VexiiRiscv supporting FP32 operations (RV32IMFC). With faster FP operations, BNRV achieves an overall speedup of up to 4.37× compared to the unextended RV32IMFC CPU. Operating at a core frequency of 500 MHz, **the BNRV extension increases the generation speed to between 10.3 and 13.3 tokens per second**. These results demonstrate the efficiency and effectiveness of the BNRV extension in accelerating the inference of BitNet-based transformers.

E. Energy Efficiency for BitNet Inference

As a comprehensive metric of power and performance, energy consumption per model inference is especially crucial for edge devices. According to the metrics evaluated in Sec. IV-B to Sec. IV-D, we calculate the energy per inference with and without the BNRV extension, as shown in Tab. VIII.

TABLE VIII: Comparison of Energy for Inference

MLP Inference			
Extension	RV32IMC	+BNRV4	+BNRV32
Energy (μJ)	3.29	1.27	0.85
Reduction	-	-61.4%	-74.2%
LeNet Inference			
Extension	RV32IMFC	+BNRV4	+BNRV32
Energy (μJ)	124.54	53.18	39.18
Reduction	-	-57.3%	-68.5%
Tiny LLaMa2 Inference			
Extension	RV32IMC	+BNRV4	+BNRV32
Energy (μJ)	441.19	183.30	144.81
Reduction	-	-58.5%	-67.2%

The BNRV significantly reduces energy consumption by 57.3% to 74.2% compared to the baseline CPU, resulting in a more energy-efficient inference of BitNet-based models.

F. Comparison with Existing SIMD Extensions

As discussed in Sec. II-D, there are some existing extensions for QNN acceleration. However, many of these extensions lack software support for deploying BitNet-based models, and some do not have open-source hardware implementations. To enable a comparison between BNRV and these existing extensions, we implemented several SIMD extensions on VexiiRiscv and developed the BitNet software support for them.

First, we implemented the SIMD 8-bit dot product (8b DotP) instruction that computes the dot product between two 8-bit vectors. This instruction is commonly used for accelerating 8-bit QNNs and is found in SIMD extensions such as Xpulpv2 [23] and ARM Neon [24]. Second, we implemented the mixed-precision 8-bit×2-bit SIMD dot product (8bx2b DotP) instruction from MPIC [14], which builds upon the 8b DotP instruction to compute the dot product between 8-bit vectors and 2-bit vectors. We compared BNRV with these two extensions on the BitNet-based MLP model and LLaMa2 model, and the results are shown in Tab. IX.

TABLE IX: Comparison between SIMD Extensions

Extension	Area (μm^2)	MLP Latency	LLaMa Latency
RV32IMC	38782 (1.00)	1.73 ms (1.00)	232.56 ms (1.00)
+8b DotP	40195 (1.04)	1.06 ms (0.61)	185.20 ms (0.80)
+8bx2b DotP	40128 (1.03)	0.67 ms (0.39)	97.09 ms (0.42)
+BNRV4	**39281 (1.01)**	0.67 ms (0.39)	97.09 ms (0.42)
+BNRV8	39666 (1.02)	0.51 ms (0.29)	80.36 ms (0.35)
+BNRV32	40274 (1.04)	**0.44 ms (0.25)**	**75.19 ms (0.32)**

The comparison results indicate that although the 8-bit dot product instruction can accelerate BitNet MatMul, the speedup is reduced due to the unpacking requirement. Conversely, the 8-bit/2-bit mixed dot product instruction achieves a speedup comparable to our BNRV4 instruction. However, it requires a larger area compared to BNRV4 since it is extended from the 8-bit multipliers.

Compared to these two extensions, BNRV4 provides a decent speedup with minimal overhead. BNRV8 further improves the speedup through greater parallelism and better bit utilization, and the largest variant, BNRV32, achieves the highest speedup overall.

V. Conclusion

We presented BNRV, a lightweight 8-bit SIMD extension along with a corresponding software library for RISC-V CPUs, tailored to accelerate BitNet inference with 1-to-2-bit quantized weights. Experimental results show that compared to the base RV32IMC CPU, BNRV achieves up to 10.95× speedup on BitNet matrix multiplication, up to 3.93× speedup on BitNet-based MLP model inference, and up to 3.11× speedup on BitNet-based language model inference. The multiplier-free hardware architecture of BNRV enables these performance enhancements while maintaining minimal overhead, with only 4% additional area and 2% increased power consumption. BNRV allows for significant model compression while maintaining acceptable accuracy, demonstrating its effectiveness for edge AI applications that require both high performance and resource efficiency.

References

[1] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *ECCV 2016*, pp. 525–542, 2016.

[2] A. Bulat and G. Tzimiropoulos, "Xnor-net++: Improved binary neural networks," *arXiv preprint arXiv:1909.13863*, 2019.

[3] S. Zhu, L. H. K. Duong, and W. Liu, "Xor-net: An efficient computation pipeline for binary neural network inference on edge devices," in *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 124–131, 2020.

[4] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, and F. Wei, "Bitnet: Scaling 1-bit transformers for large language models," *arXiv preprint arXiv:2310.11453*, 2023.

[5] S. Ma and et al., "The era of 1-bit llms: All large language models are in 1.58 bits," *arXiv preprint arXiv:2402.17764*, vol. 1, no. 4, 2024.

[6] B. Moons, D. Bankman, L. Yang, B. Murmann, and M. Verhelst, "Binareye: An always-on energy-accuracy-scalable binary cnn processor with all memory on chip in 28nm cmos," in *2018 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–4, 2018.

[7] Y.-C. Lo, Y.-C. Kuo, and et al., "Physically tightly coupled, logically loosely coupled, near-memory bnn accelerator (ptll-bnn)," in *ESSCIRC 2019*, pp. 241–244, 2019.

[8] R. Andri, G. Karunaratne, L. Cavigelli, and L. Benini, "Chewbaccann: A flexible 223 tops/w bnn accelerator," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2021.

[9] A. Vaswani, N. Shazeer, and et al., "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[10] R. Eldan and Y. Li, "Tinystories: How small can language models be and still speak coherent english?," *arXiv preprint arXiv:2305.07759*, 2023.

[11] R.-V. Foundation, "Risc-v "p" standard extension for packed-simd instructions." https://five-embeddev.com/riscv-user-isa-manual/riscv-user-2.2/p.html, June 2024.

[12] A. Garofalo, G. Tagliavini, F. Conti, D. Rossi, and L. Benini, "Xpulpnn: Accelerating quantized neural networks on risc-v processors through isa extensions," in *DATE 2020*, pp. 186–191, 2020.

[13] G. Rutishauser, J. Mihali, M. Scherer, and L. Bonini, "xtern: Energy-efficient ternary neural network inference on risc-v-based edge systems," in *ASAP 2024*, pp. 206–213, 2024.

[14] G. Ottavi, A. Garofalo, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, "A Mixed-Precision RISC-V Processor for Extreme-Edge DNN Inference," in *ISVLSI 2020*, pp. 512–517, July 2020.

[15] O. Muller, A. Prost-Boucle, A. Bourge, and F. Pétrot, "Efficient decompression of binary encoded balanced ternary sequences," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1962–1966, 2019.

[16] T. Chen, T. Moreau, and et al., "TVM: An automated End-to-End optimizing compiler for deep learning," in *OSDI 18*, (Carlsbad, CA), pp. 578–594, 2018.

[17] A. Burrello and et al., "Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus," *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1253–1268, 2021.

[18] SpinalHDL, "SpinalHDL/VexiiRiscv." github.com/SpinalHDL/VexiiRiscv, June 2024.

[19] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] H. Touvron and et al., "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[22] A. Karpathy, "llama2.c: Inference llama 2 in one file of pure c." github.com/karpathy/llama2.c, June 2024.

[23] M. Gautschi and et al., "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[24] Arm, "Arm neon®." www.arm.com/technologies/neon, June 2024.