# Mixed Precision Quantization for ReRAM-based DNN Inference Accelerators

Sitao Huang[1], Aayush Ankit[2], Plinio Silveira[3], Rodrigo Antunes[3], Sai Rahul Chalamalasetti[4], Izzat El Hajj[5], Dong Eun Kim[2], Glaucimar Aguiar[3], Pedro Bruel[4,6], Sergey Serebryakov[4], Cong Xu[4], Can Li[4] Paolo Faraboschi[4], John Paul Strachan[4], Deming Chen[1], Kaushik Roy[2], Wen-mei Hwu[1], and Dejan Milojicic[4]
[1]University of Illinois at Urbana-Champaign, USA [2]Purdue University, USA [3]Hewlett Packard Enterprise, Brazil
[4]Hewlett Packard Enterprise, USA [5]American University of Beirut, Lebanon [6]University of São Paulo, Brazil
{shuang91,dchen,w-hwu}@illinois.edu, {aankit,kim2976,kaushik}@purdue.edu, izzat.elhajj@aub.edu.lb, {firstname.lastname}@hpe.com

## ABSTRACT

ReRAM-based accelerators have shown great potential for accelerating DNN inference because ReRAM crossbars can perform analog matrix-vector multiplication operations with low latency and energy consumption. However, these crossbars require the use of ADCs which constitute a significant fraction of the cost of MVM operations. The overhead of ADCs can be mitigated via partial sum quantization. However, prior quantization flows for DNN inference accelerators do not consider partial sum quantization which is not highly relevant to traditional digital architectures. To address this issue, we propose a mixed precision quantization scheme for ReRAM-based DNN inference accelerators where weight quantization, input quantization, and partial sum quantization are jointly applied for each DNN layer. We also propose an automated quantization flow powered by deep reinforcement learning to search for the best quantization configuration in the large design space. Our evaluation shows that the proposed mixed precision quantization scheme and quantization flow reduce inference latency and energy consumption by up to 3.89× and 4.84×, respectively, while only losing 1.18% in DNN inference accuracy.

## KEYWORDS

Mixed precision quantization, ReRAM, DNN inference accelerators

## 1 INTRODUCTION

Quantization is an important optimization for reducing DNN inference latency and energy consumption [1–10]. Workflows that apply quantization to DNNs commonly target the weights and inputs of the DNN layers. Reducing the number of bits used to represent weights and inputs saves memory space, reduces data movement overhead, and shortens the latency of arithmetic operations. When different quantization configurations are chosen for different layers, the quantization is considered to be *mixed precision*.

ReRAM-based accelerators have shown great potential for accelerating DNN inference because ReRAM crossbars can perform analog matrix-vector multiplication (MVM) operations with low latency and energy consumption [11–15]. However, ReRAM crossbars require ADCs to convert the partial sum computed by each crossbar from an analog to a digital value before it is combined with partial sums from other crossbars. These ADCs consume a large fraction of the total chip area and energy. Since the cost of ADCs scales exponentially with their precision, reducing the precision of ADCs is an important optimization. Hence, ReRAM-based accelerators provide the opportunity for a third important quantization target, the partial sums at the ADC output, which are not usually a concern for traditional digital architectures.

To take advantage of this opportunity, we propose an automated mixed precision quantization flow that jointly targets weights, inputs, and partial sums. Since the design space is large and prohibitive to search exhaustively, we use deep reinforcement learning (DRL) to search for the best configuration. Our evaluation shows that the proposed quantization flow reduces inference latency and energy consumption by up to 3.89× and 4.84×, respectively, while only losing 1.18% in accuracy.

We make the following contributions:

- A quantization scheme for ReRAM-based DNN inference accelerators that jointly targets weights, inputs, and partial sums, with a functional simulator that models the quantization scheme
- An automated mixed precision quantization flow powered by deep reinforcement learning that searches for the best quantization configuration for DNN inference on ReRAM-based accelerators
- An evaluation of the joint impact of weight, input, and partial sum quantization on the energy and latency of ReRAM-based DNN inference accelerators

## 2 QUANTIZATION SCHEME

### 2.1 Background

ReRAM crossbars are circuits capable of performing MVM operations with low latency and energy consumption by leveraging analog computing. Figure 1 shows a high-level diagram of a typical crossbar architecture. The weights of a matrix are stored in the resistive memory cells of the crossbar. $g_{ij}$ in Figure 1 is the conductance of a memory cell. The input vector is applied as a voltage at the rows of the crossbar ($V_i$). The output vector is read as the current at the columns of the crossbar ($I_j$). The current is then converted to a digital value using an ADC.

Since practical crossbars are only capable of performing low precision MVM operations, higher precision MVM operations are


*ASPDAC '21, January 18–21, 2021, Tokyo, Japan*

ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431554

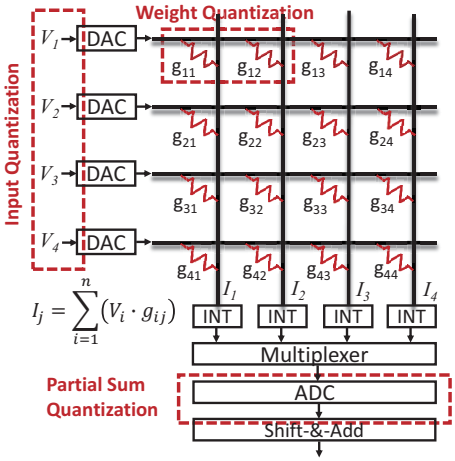


**Figure 1: ReRAM Crossbar Architecture and Quantization**

realized by bit-slicing the weights and inputs. Weight slices are distributed across multiple crossbars, and the partial sums of each crossbar are then shifted and added together. Input slices are streamed sequentially into each crossbar, and the partial sums of each input slice are also shifted and added together. If the weight matrix dimensions are larger than the crossbar dimensions, the matrix is divided into tiles and the partial sums of each tile are then added to produce the final MVM result.

## 2.2 Weight and Input Quantization

Recent research has revealed that weights (synapses) and inputs (activations) in a DNN typically do not need full precision to guarantee the DNN prediction accuracy [3, 5]. In general, using a low bit-width format to represent weights and inputs saves memory space, reduces data movement overhead, and shortens the computation latency. Weight and input quantization have been thoroughly studied for traditional architectures. In this work, we propose weight and input quantization schemes for ReRAM crossbars.

Weight quantization in the crossbar architecture can be achieved by either changing the number of crossbars or the number of bits per crossbar cell. However, the impact of device-circuit non-idealities in the crossbar (both linear and non-linear) increases with increasing bits per crossbar cell leading to significant losses in network accuracy [16]. Hence, this work assumes a fixed two bits of weights are stored in each crossbar cell [13] and implements weight quantization by varying the number of crossbars used to store the weights.

Input quantization in the crossbar architecture can be obtained by either changing the number of input slices streamed or the number of bits per slice. However, increasing the bits per input slice requires increasing the ADC precision which increases the overhead of the ADC non-linearly. Hence, this work assumes a fixed one bit per input slice [13] and implements input quantization by varying the number of input slices streamed.

## 2.3 Partial Sum Quantization

While weight and input quantization have received significant attention by quantization flows because of their relevance to digital architectures, partial sum quantization has not been thoroughly

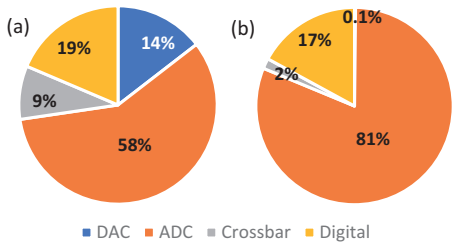


**Figure 2: Distribution of MVM Cost: (a) Energy Distribution (b) Area Distribution**

**Table 1: Energy savings of reduced ADC resolution**

| ADC Res. | LSTM (24 tiles) | | MLP (9 tiles) | |
|---|---|---|---|---|
| | Energy ($\mu$J) | Diff. | Energy ($\mu$J) | Diff. |
| **8 (baseline)** | 59.0 | - | 16.6 | - |
| **4** | 46.3 | -21.4% | 13.2 | -20.6% |
| **2** | 41.0 | -30.5% | 11.6 | -30.3% |
| **1** | 38.3 | -35.0% | 10.8 | -35.1% |

studied. Partial sum quantization in the crossbar architecture can be achieved by reducing the precision of ADCs at crossbar outputs. We model the impact of partial sum quantization combined with weight and input quantization by implementing a functional simulator described in Section 4.2. We show that partial sum quantization can substantially reduce the energy consumption and latency of DNN inference accelerators.

***Energy.*** Figure 2(a) shows the distribution of energy consumption across four components (DAC, ADC, crossbar, and digital peripherals) for a 16-bit MVM operation [13]. It is clear that ADCs are the dominating component of the total energy consumption. Consequently, quantizing partial sums by operating ADCs at lower resolutions can yield significant energy reductions in the MVM operation. Table 1 shows how reducing ADC resolution translates to reductions on the total inference energy consumption. These results assume that ADC power decreases linearly with the ADC resolution, which is a conservative assumption.

***Latency.*** The latency of MVM operations is limited by ADCs, but to understand why, it is important to first look at ADC area. Figure 2(b) shows that ADCs consume a substantial amount of the area in the crossbar architecture. For this reason, ADCs are time-multiplexed such that one ADC is reused across all the columns of the crossbar [13]. The typical size of a crossbar is 128 × 128. Consequently, even with an ADC working at very high sampling frequency such as 1GHz, a crossbar read operation requires 128 ns, while typical crossbar reads without ADC require 5-20 ns [17]. Area consumption and sampling time for an ADC decrease with reducing ADC resolution. Consequently, partial sum quantization can achieve significant reductions in MVM latency.

# 3 MIXED PRECISON QUANTIZATION FLOW

We model the mixed precision quantization problem as a reinforcement learning (RL) problem. In an RL problem, an *agent* (search engine) interacts with the *environment* and learns the best *policy* to take *actions* in certain states of the environment. The environment in an RL problem can be modeled with a Markov Decision Process

**Table 2: Tunable parameters**

| Parameters | Values |
|---|---|
| Weight bits | 4, 8, 16, 32 |
| Weight bits (fractional) | 1, 2, ..., Weight bits - 1 |
| Input bits | 4, 8, 16, 32 |
| Input bits (fractional) | 1, 2, ..., Input bits - 1 |
| ADC precision | 1,2,3,4,5,6,7,8 |

(MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition function that describle the dynamics of the MDP system, $\mathcal{R}$ is the reward function that maps state and action pair to a real-valued reward number, $\gamma$ is the discount factor that describes the degradation of future rewards.

## 3.1 MDP Modeling

In this quantization problem, we define the MDP as follows. State space $\mathcal{S}$ is defined as all possible configurations of the DNN. We can see this is a huge state space. $O(|\mathcal{S}|)$ is exponential to the number of layers in the DNN. Action space $\mathcal{A}$ is defined as all possible quantization configurations for a layer in the DNN. Transition function $\mathcal{P}$ is defined in a way such that we quantize the DNN layer by layer from the first layer. After an action is applied on the current layer, the environment moves to the next layer. Reward $\mathcal{R}$ is defined so that accuracy, power, and latency of the quantized DNN are all captured. Note that reward is a function of both current state and current action. We will discuss more about the reward definition in the following paragraphs. Discount factor $\gamma$ is set to 1. This is based on the finite-horizon problem setting, reward setting, and the fact that we only care about the final accuracy of the fully quantized DNN. In the RL setting, a policy $\pi$ is a function that maps state space to action space. In other words, a policy tells the agent the action $a$ to take given the current state $s$: $a = \pi(s)$.

## 3.2 Action Definition

The actions are defined as quantization configurations for a layer in the DNN, which includes weight quantization, input quantization, and ADC precision. All the tunable parameters are listed in Table 2. There are two parameters to describe quantizaiton of each of weights and inputs, one is the total bit width, the other is the bit width for the fractional part. The possible values of total bits are powers of 2. This is a constraint that comes from bit slicing in the functional simulator. The bit width of the fractional parts can be any number less than the total bit width. ADC precision can be any integer between 1 and 8. Each action represents a configuration of these parameters for one DNN layer.

## 3.3 Reward Assignment

We define reward $R(s, a)$ at state $s$, action $a$ as a function of accuracy cost and hardware cost (energy and latency) of the quantized DNN when running on ReRAM accelerators.

To capture the inference error of quantized DNNs, instead of using prediction accuracy of quantized DNN as in previous works [3, 5, 6], we use the following definition of the error of quantized model:

$$Cost_{\text{accuracy}} = Loss_{\text{quantization}} - Loss_{\text{original}} \tag{1}$$

where $loss_{\text{quantization}}$ and $loss_{\text{original}}$ are the cross-entropy losses of the quantized DNN model and the original model respectively. The intuition is to use original model as the reference and any differences in the output are counted as errors introduced by quantization. The major reason of using loss instead of accuracy is to reduce the number of inferences during the evaluation of a quantization scheme and speed up the search progress. To model the hardware cost (energy and latency) of quantization schemes, we use the fractions of bitwidth over original bitwidth in each layer, weighted by the number of weights and inputs in DNN layers as well as ADC bitwidth. That is, the hardware cost of a quantization scheme can be approximated as

$$Cost_{\text{hardware}} = \sum_i \alpha^{B^i_{\text{ADC}}} \left( f^i_{\text{input}} \frac{B^i_{\text{input}}}{B_{\text{full}}} + f^i_{\text{weight}} \frac{B^i_{\text{weight}}}{B_{\text{full}}} \right) \tag{2}$$

where $B^i_{\text{ADC}}$, $B^i_{\text{input}}$, and $B^i_{\text{weight}}$ are the ADC bitwidth, input bitwidth, and weight bitwidth of the $i^{\text{th}}$ layer respectively. $B_{\text{full}}$ is the full bitwidth without quantization for inputs and weights. $f^i_{\text{input}}$ and $f^i_{\text{weight}}$ are the fractions of number of inputs and number of weights over total inputs and total weights respectively.

Reward is calculated based on both costs as follows:

$$Reward = -T(Cost_{\text{accuracy}}) - Cost_{\text{hardware}} \tag{3}$$

where $T$ is a threshold function: $T_t(x) = \infty \cdot \mathbb{1}_{x>t} + x$. Here $\mathbb{1}_{x>t}$ is the indicator function which equals to 1 when $x > t$ and 0 otherwise. $t$ is the threshold.

## 3.4 Learning Algorithm

With the definition of MDP, our goal is to find an optimal policy that gives us the largest expected reward for the starting state (the whole DNN for quantization). In order to find out the optimal policy, we need some estimation on the potential of each state and each action, with current policy $\pi$, i.e., the expected value of state $s$ or state-action pair $(s, a)$. The expected value of state is typically called state-value funciton $V^\pi(s)$, while the expected value of state-action pair is called Q-value function $Q^\pi(s, a)$.

One thing to notice is that both state space and action space are very large, and its almost impossible to search for optimal policy with brute-force. We parameterize the policy as $\pi_\theta$ where $\theta$ is the parameter. With the parameterized policy, we are able to apply policy gradient algorithms to the problem, which is a family of RL algorithms widely used in practice.

The basic policy gradient theorem states that the gradient of the value function can be expressed as

$$\nabla v^\pi = \frac{1}{1-\gamma} \mathbb{E}_{s\sim\eta^\pi, a\sim\pi(s)} \left[ Q^\pi(s, a) \nabla \log \pi(a|s) \right] \tag{4}$$

where $Q^\pi(s, a)$ is the expected value of state $s$ and action $a$, which can be estimated with value-based methods in RL. Value-based methods learn the value function, rather than the optimal policy itself. $\eta^\pi$ is the estimated normalized state occupancy under policy $\pi$. This formula essentially states the weighted average of values over potential trajectories of the agent, and the weights are the

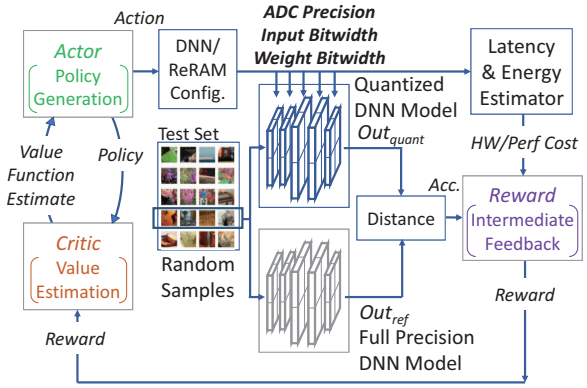


**Figure 3: Reinforcement Learning based Mixed-Precision Quantization Flow**

function of policy gradient. The policy gradient algorithms work as follows. First of all, we start with some policy with random parameters $\theta$. We use value-based algorithms to evaluate the expected values of state-action pairs $(s, a)$. Then we calculate the gradient of $\theta$ using the formula, and we can update $\theta$ and get a new policy $\pi'$. With the new policy, we can again evaluate the execpted value of each state-action pair $(s, a)$ again. This completes one iteration of the algorithm. We can see the iteration consists of two parts: (1) "Actor": update $\theta$ with gradients of $\theta$, and propose a new policy $\pi'$; (2) "Critic": evaluate the current policy $\pi$ with value-based methods. Therefore, this type of combination of policy gradient and value-based method is often called "actor-critic" methods. In this work, the actor and critic are both implemented with fully connected DNNs.

## 3.5 Putting It All Together

With definition of all the details of MDP, we can put together everything to form a complete optimization flow. The complete flow is illustrated in Figure 3.

The optimization flow is a combination of reinforcement learining components and quantized model evaluation components.

**RL components.** The RL components consist of three parts: *actor*, *critic*, and *reward*. The *actor* part generates new policies based on the value function estimates by calculating gradients of parameters. The generated policy is parsed by DNN/ReRAM configurator which generates ADC precision and all the bitwidths for each layer in the DNN. The *critic* part reads reward from the environment and calcuates Q-value function of this MDP. The *reward* part takes in the loss measure collected from quantized model and original model, as well as the DNN configuration, and calculates the reward for current state $s$ and action $a$.

**Quantized model evaluation components.** The evaluation components consist of *test dataset*, *quantized DNN model*, *full precision DNN model*, and *loss measure*. *Test dataset* generates batches of random test samples. The batch size is configurable. *Quantized DNN model* and *full precision DNN model* take in the sample batch from test dataset, and run model inference. Most likely the full precision DNN model will generate a smaller loss $loss_{\text{Ref}}$ than that of the quantized model $loss_{\text{Quant}}$. The flow calculates the differences $d = loss_{\text{Quant}} - loss_{\text{Ref}}$, and use $4^d$ as the cost for the accuracy drop.

# 4 METHODOLOGY

## 4.1 Performance Simulation

To evaluate the impact of different quantization schemes on DNN inference energy consumption and latency, we use the PUMA [15] simulator, PUMAsim, which is a cycle-level architecture simulator for ReRAM-based accelerators. PUMAsim runs applications compiled with the PUMA compiler and provides detailed traces of execution. The simulator incorporates timing and power models of all system components. The simulator provides options for configuring various architectural parameters, including input and weight bit-width. We extend the simulator to also configure ADC resolution for evaluating partial sum quantization. We consider 2-bits per device for weight slicing and 1-bit per slice for input slicing, which are the typical parameters used in past crossbar-based accelerators [13, 15].

## 4.2 Functional Simulation

To evaluate the impact of different quantization schemes on DNN inference accuracy, we develop a functional simulator to simulate the arithmetic behavior of ReRAM-based accelerators which PUMAsim does not capture. Although several libraries such as Distiller [18], Model Optimization Toolkit [19], etc. have been developed using TensorFlow and PyTorch to enable software and hardware co-design studies for quantization, such frameworks cannot emulate the precise implication of quantization on ReRAM-based accelerators because of the intrinsic differences between digital and ReRAM-based hardware. Digital accelerators typically express layer operations as general matrix-matrix multiplication, which use floating or fixed point computation units. On the other hand, ReRAM-based accelerators typically express layer operations as a tiled MVM which use bit-serial computation units operating in the analog domain (discussed in Section 2). For this reason, we develop our own functional simulator using PyTorch to analyze the impact of quantization of different aspects of crossbar hardware: weights, inputs, and partial sums. The functional simulator models the key phases of MVM computation in typical crossbar accelerators, namely iterative MVM, tiling, and bit-slicing, and ignores the memory and communication aspects which are captured by PUMAsim.

## 4.3 Search Flow

We use the functional simulator to guide our search flow, and the performance simulator to calibrate our hardware cost model and evaluate the search result. We use prediction error instead of model accuracy as a metric to evaluate quantization accuracy. Prediction error requires fewer samples to estimate compared to model accuracy. We extract a small batch of random samples from the whole dataset, run the quantized DNN model with the functional simulator, and compare its outputs with outputs from full-precision DNN model to get the predicion error.

# 5 EVALUATION

## 5.1 Benefits of Mixed Precision Quantization

We use the performance simulator to evaluate the benefits of mixed precision quantization in ReRAM-based accelerators. We simulate

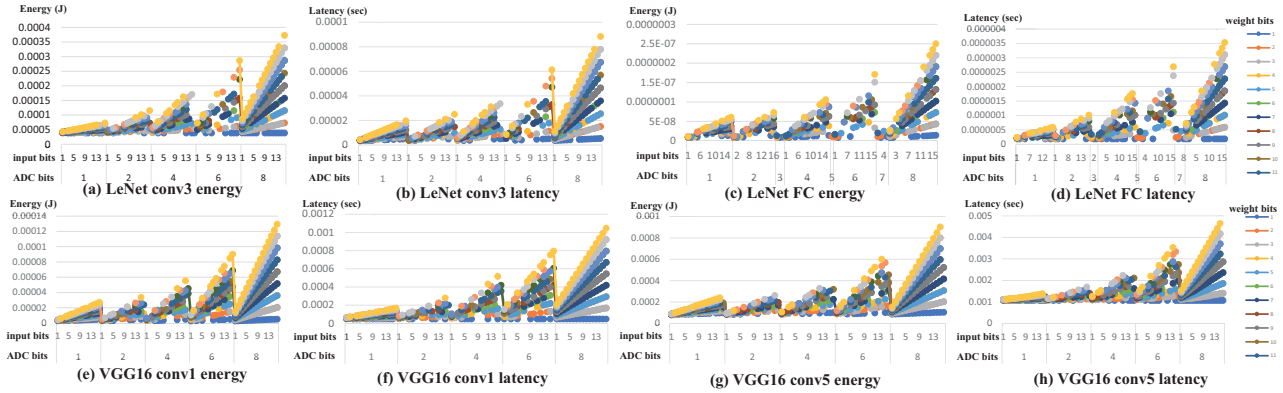


**Figure 4: Energy and Latency under Quantization**

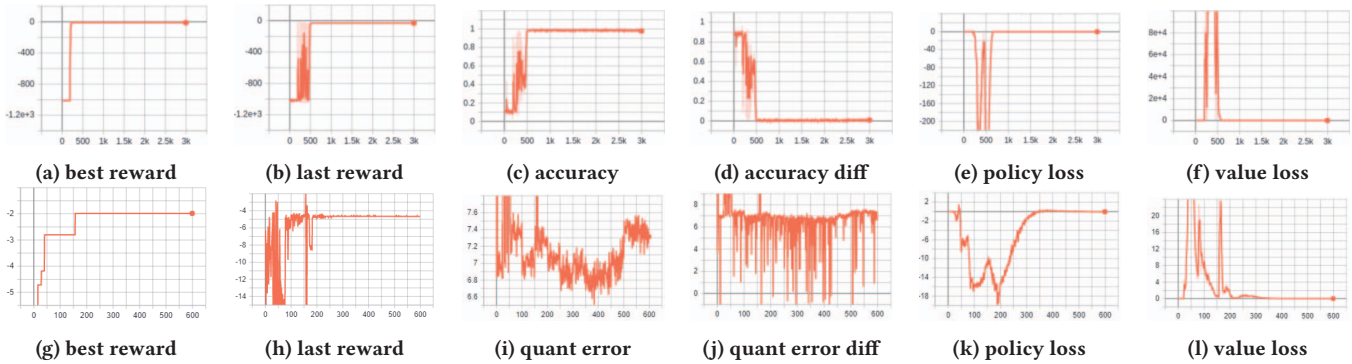


**Figure 5: Intermediate Values in Search (top: LeNet; buttom: VGG16)**

all the combinations of quantization configurations of each layer in LeNet and VGG16, and collect energy and latency results of the simulated layer. Figures 4(a)-(d) show the energy and latency numbers of the third convolution layer and the fully connected layer in LeNet. Figures 4(e)-(h) show the energy and latency numbers of the first convolution layer and the fifth convolution layer in VGG16. As we can see from these figures, energy and latency numbers follow similar trends as bit-widths change. With the same ADC precision, energy and latency change linearly as input bitwidth and weight bitwidth change. With higher ADC precision, the energy and latency are more sensitive to changes in input bitwidth and weight bitwidth. Although Figure 4 shows regular patterns with a fixed ADC precision, the design space of weight bitwidth and input bitwidth combined with ADC precision is non-linear.

## 5.2 Quantization Search Flow

We use our proposed mixed precision quantization search flow to search for the optimal quantization schemes for LeNet and VGG16 that have lowest energy and latency while not losing accuracy much. The intermediate reward and loss during the searches are shown in Figure 5. Figures 5(a)-(f) show the results for LeNet search, while Figures 5(g)-(l) show the results for VGG search. In our experiments, we ran LeNet search for 3,000 episodes, and VGG16 search for 600 episodes. However, note that LeNet search also converges within 600 episodes. As shown in the figure, search converges in both cases. At the beginning of the search, the RL agent takes random actions to explore the design space (*environment*). The agent learns about the environment at the same time. After warming up, the agent starts to apply the knowledge it learned into the search, and optimizes policy to improve the expected future rewards. At the end, the agent reports the best policy it discovered in the search, which translates to the best quantization configuration discovered. Table 3 lists the energy, latency, and accuracy of a few quantization schemes for LeNet discovered by the search flow. $Q_{base}$ is the baseline with full bit-widths. Each $(i, w, a)$ tuple describes the bitwidths for inputs ($i$), weights ($w$), and ADC ($a$) for a layer. The four tuples correspond to quantization schemes for conv1, conv2, conv3, and fully connected layers in LeNet respectively. As the table shows, the discovered quantization scheme achieves up to 4.84× savings in energy and 3.89× savings in latency while only losing 1.18% accuracy, compared to $Q_{base}$. Schemes with even higher accuracy but less latency and energy savings are also discovered, e.g. $Q_c$. These design points reflect the tradeoffs between accuracy and performance (or resource), and provide different design options for different design requirements. We also evaluate the search results for VGG16 and we get similar energy and latency savings for VGG16.

## 6 RELATED WORK

Various works apply quantization to DNNs to improve the efficiency of their execution [1–10]. ADMM-NN [20] is a framework that

**Table 3: LeNet quantization schemes**

| Quantization | $E$ ($\mu$J) | $T$ (ms) | Accuracy |
|---|---|---|---|
| $Q_{base}$ | 850.99 (1.00×) | 2.95 (1.00×) | 97.27% (-0.00%) |
| $Q_A$ | 175.61 **(4.84×)** | 0.76 **(3.89×)** | 96.09% **(-1.18%)** |
| $Q_B$ | 229.82 (3.70×) | 0.85 (3.48×) | 96.29% (-0.98%) |
| $Q_C$ | 468.48 (1.82×) | 1.69 (1.74×) | 97.07% (-0.20%) |

NOTE: Values in parentheses are savings compared to $Q_{base}$.
$Q_A$: (4, 16, 7), (4, 8, 8), (4, 8, 7), (4, 16, 8); $Q_B$: (16, 8, 8), (4, 8, 8), (8, 8, 8), (4, 8, 8);
$Q_C$: (16,16,6), (16,8,8), (4,8,7), (4,16,7); $Q_{base}$: (16,16,8),(16,16,8),(16,16,8),(16,16,8).

performs quantization and pruning jointly. Ares [21] is a framework for quantifying the resilience of DNNs to faults, and considers the impact of different quantization schemes on resilience. Choi et al. [22] reduce the mismatch between forward and backward passes when training networks that use quantization. Sakr et al. [23] propose quantization for back-propagation, not just inference. All these works focus on quantization of DNNs in general without architecture-specific considerations.

Bit fusion [24] and UNPU [25] provide architecture support for dynamically reconfiguring bit width in DNN accelerators to support different levels of quantization. OLAccel [26] is an accelerator that enables better quality quantization by handling outliers separately. Various works that focus on quantization have also targeted FPGAs, such as REQ-YOLO [27] which focuses on FPGA resource awareness, and FINN [28] which specializes in binarized neural networks. All these works focus on digital accelerators, whereas our work focuses on ReRAM-based accelerators and the unique opportunities they provide.

Zhu et al. [29] provide a framework for quantizing CNNs on single-bit ReRAM crossbars. Zhang et al. [30] provide design guidelines for ReRAM-based DNN accelerators and include the impact of ADC quantization as part of their study. Our framework performs joint quantization of both weights and partial sums on two-bit crossbars based on deep reinforcement learning.

Various works propose ReRAM-based accelerators for DNN inference [11–15] and training [31–35]. Our work proposes a framework for quantization of DNNs to configure such accelerators. Other frameworks have also been proposed for transforming [36, 37] and pruning [38] DNNs for such accelerators.

## 7 CONCLUSION

We present a mixed precision quantization scheme and an automated quantization flow for optimizing DNN inference on ReRAM-based accelerators. The flow uses deep reinforcement learning to find the best configuration of weight quantization, input quantization, and partial sum quantization across DNN layers. The evaluation shows that the quantization scheme enables more optimization opportunity and the automated quantization flow can effectively search for the best quantization configuration. The quantization configuration discovered by the search flow achives up to 3.89× and 4.84× improvement over baseline without quantization in terms of inference latency and energy respectively, while only losing 1.18% in DNN inference accuracy.

## REFERENCES


[1] Patrick Judd et al. Reduced-precision strategies for bounded memory in deep neural nets. *arXiv preprint arXiv:1511.05236*, 2015.
[2] Bert Moons et al. Energy-efficient convnets through approximate computing. In *WACV*, pages 1–8. IEEE, 2016.
[3] Darryl Lin et al. Fixed point quantization of deep convolutional networks. In *ICML*, pages 2849–2858, 2016.
[4] Charbel Sakr et al. Analytical guarantees on numerical precision of deep neural networks. In *ICML*, pages 3007–3016. JMLR. org, 2017.
[5] Lu Hou et al. Loss-aware weight quantization of deep networks. *arXiv preprint arXiv:1802.08635*, 2018.
[6] Kuan Wang et al. HAQ: Hardware-aware automated quantization with mixed precision. In *CVPR*, pages 8612–8620, 2019.
[7] Junsong Wang et al. Design flow of accelerating hybrid extremely low bit-width neural network in embedded FPGA. In *FPL*, pages 163–169, 2018.
[8] Cong Hao et al. Fpga/dnn co-design: An efficient design methodology for iot intelligence on the edge. In *DAC*, New York, NY, USA, 2019.
[9] Yuhong Li et al. Edd: Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions. In *DAC*. IEEE Press, 2020.
[10] Cheng Gong et al. VecQ: Minimal loss dnn model compression with vectorized weight quantization. *IEEE Transactions on Computers*, (01):1–1, may 5555.
[11] Xiaoxiao Liu et al. RENO: A high-efficient reconfigurable neuromorphic computing accelerator design. In *DAC*, pages 1–6. IEEE, 2015.
[12] Ping Chi et al. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *ISCA*, 2016.
[13] Ali Shafiee et al. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *ISCA*, ISCA'16, pages 14–26. IEEE Press, 2016.
[14] Ben Feinberg et al. Making memristive neural network accelerators reliable. In *HPCA*, pages 52–65. IEEE, 2018.
[15] Aayush Ankit et al. PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *ASPLOS*, pages 715–731, 2019.
[16] Indranil Chakraborty et al. GENIEx: A Generalized Approach to Emulating Non-Idealities in Memristive X-bars Using Neural Networks. In *DAC*, 2020.
[17] Matthew J Marinella et al. Multiscale co-design analysis of energy, latency, area, and accuracy of a ReRAM analog neural training accelerator. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(1):86–101, 2018.
[18] Neta Zmora et al. Neural network distiller, June 2018.
[19] TensorFlow. Model optimization toolkit.
[20] Ao Ren et al. ADMM-NN: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. In *ASPLOS*, 2019.
[21] Brandon Reagen et al. Ares: A framework for quantifying the resilience of deep neural networks. In *DAC*, pages 1–6. IEEE, 2018.
[22] Yoojin Choi et al. Learning low precision deep neural networks through regularization. *arXiv preprint arXiv:1809.00095*, 2018.
[23] Charbel Sakr et al. Per-tensor fixed-point quantization of the back-propagation algorithm. In *ICLR*, 2019.
[24] Hardik Sharma et al. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *ISCA*, pages 764–775, 2018.
[25] Jinmook Lee et al. UNPU: A 50.6 TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision. In *ISSCC*, pages 218–220, 2018.
[26] Eunhyeok Park et al. Energy-efficient neural network accelerator based on outlier-aware low-precision computation. In *ISCA*, pages 688–698. IEEE, 2018.
[27] Caiwen Ding et al. REQ-YOLO: A resource-aware, efficient quantization framework for object detection on fpgas. In *FPGA*, pages 33–42, 2019.
[28] Yaman Umuroglu et al. FINN: A framework for fast, scalable binarized neural network inference. In *FPGA*, pages 65–74, 2017.
[29] Zhenhua Zhu et al. A configurable multi-precision cnn computing framework based on single bit rram. In *DAC*, pages 1–6. IEEE, 2019.
[30] Wenqiang Zhang et al. Design guidelines of rram based neural-processing-unit: A joint device-circuit-algorithm analysis. In *DAC*, pages 1–6. IEEE, 2019.
[31] Mahdi Nazm Bojnordi et al. Memristive Boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning. In *HPCA*, 2016.
[32] Ming Cheng et al. Time: A training-in-memory architecture for memristor-based deep neural networks. In *DAC*, page 26. ACM, 2017.
[33] Linghao Song et al. Pipelayer: A pipelined ReRAM-based accelerator for deep learning. In *HPCA*, pages 541–552. IEEE, 2017.
[34] Fan Chen et al. ReGAN: A pipelined ReRAM-based accelerator for generative adversarial networks. In *ASP-DAC*, pages 178–183. IEEE, 2018.
[35] Aayush Ankit et al. PANTHER: A programmable architecture for neural network training harnessing energy-efficient reram. *IEEE Transactions on Computers*, 2020.
[36] Yu Ji et al. NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints. In *MICRO*, page 21. IEEE Press, 2016.
[37] Yu Ji et al. Bridge the gap between neural networks and neuromorphic hardware with a neural network compiler. In *ASPLOS*, pages 448–460. ACM, 2018.
[38] Yandan Wang et al. Group Scissor: Scaling Neuromorphic Computing Design to Large Neural Networks. In *DAC*, page 85. ACM, 2017.